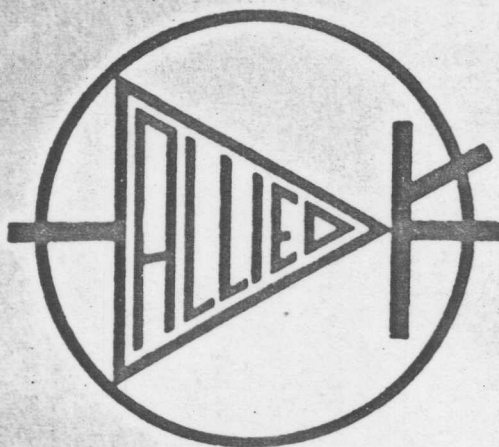


AT&AE



"LEADERSHIP IN ELECTRONICS"

OPERATING & INSTALLATION INSTRUCTIONS

FOR

ALLIED
BINARY ARITHMETIC
SUBROUTINES FOR RCA
COSMAC MICROPROCESSORS

MANUFACTURED BY:

Allied Electric (Pty) Ltd.

TELEPHONE 892-1001 TELEX 8-7823, CABLES "SOLIDSTATE" DUNSWART,
P.O. BOX 6090, DUNSWART 1508, SOUTH AFRICA
VAN DYK RD., BOKSBURG EAST. SOUTH AFRICA.

MAJOR PRODUCTS:

SILICON DIODES, THYRISTORS, TRANSISTORS AND INTEGRATED CIRCUITS

ELECTRONIC COMPONENTS INCLUDING CAPACITORS

STUDENTS' DEMONSTRATION EQUIPMENT

ELECTRIC CABLES

ELECTRIC VEHICLE BATTERY CHARGERS

GARAGE & HOME BATTERY CHARGERS, PABX & TELEPHONE EXCHANGE BATTERY CHARGERS

UNDERGROUND LOCOMOTIVE BATTERY CHARGERS

MINING WATER LEVEL CONTROL SYSTEMS INCLUDING MOSAIC PANELS

SILICON ELECTROLYTIC RECTIFIER EQUIPMENT

SILICON 3,000 VOLT DC SUBSTATIONS FOR RAILWAYS

SILICON TRACTION RECTIFIER SETS

A.C. AND D.C. ELECTRIC WELDING MACHINES

INVERTERS

THYRISTOR CONTROLLED VARIABLE SPEED D.C. MOTOR DRIVES UP TO 600 H.P.

ELECTROPLATING RECTIFIER EQUIPMENT

A.C. TO D.C. CONVERSION SETS FOR MAGNETS, CIRCUIT BREAKERS, ETC.

UNDERGROUND MINING AND RAILWAY SIGNALLING SYSTEMS

TRANSFORMERS AND TRANSDUCTORS

ELECTROSTATIC PRECIPITATORS

EMERGENCY LIGHTING EQUIPMENTS

BATTERY ELIMINATORS AND REGULATED D.C. POWER PACKS

MERCURY ARC AND SELENIUM REPLACEMENT RECTIFIERS

ELECTRONIC CONTROL EQUIPMENT

TELEMETRY SYSTEMS

SOLE SOUTH AFRICAN DISTRIBUTORS FOR:

Claude Lyons Ltd. (U.K.) — Regulac variable transformers, regulated power supplies, voltage stabilisers, constant voltage transformers.

Fairchild Automation (U.S.A.) — Integrated circuit and transistor testers.

Westinghouse Electric Corp. (U.S.A.) — Semi-conductors, panel instruments, P.C. Board materials, T.V. picture tubes.

Nife Batteries (Jungner) — Emergency lighting systems, Nife nickel-cadmium batteries, sealed cells, inverters.

Emerson Electric (U.S.A. & U.K.) — Variable frequency drives, uninterruptable power supplies up to 2000 KVA, inverters.

LTT (France) — Microwave components, tantalum capacitors, semi-conductors, ferrite materials.

Signetics (U.S.A.) — Integrated circuits of all types.

Hivac/Vitality (U.K.) — Lamps - switchboard, signalling. P.O. ballast lamps, neon indicator lamps.

General Telephone & Electronics (GTE - U.S.A.) — Turn-key microwave and radio communication projects.
Telephone systems, Supervisory, telemetry and remote control systems.

RCA Solid State (U.S.A.) — Integrated circuits, Power transistors, rectifiers, thyristors, triacs, diodes, transistor arrays, R.F. Power transistors.

RCA Electronic Components (U.S.A.) — Industrial and military tubes, test instruments, optical devices, photo-diodes, CRT displays, black and white TV picture tubes, Transmitter/Receiver valves.

Mawdsley's Ltd. (U.K.) — Thyristor variable speed drives to 1000 h.p. Students electrical demonstration equipment, D.C. motors, traction motors, computer standby power supplies.

Cableform (U.K.) — Electric vehicle controllers, cable connectors, contactors, loco and fork truck plugs and sockets.

Lucas Electrical (U.K.) — Press fit diodes, zener diodes, avalanche P.O. diodes, bridge rectifiers.

Louis Allis Corporation (U.S.A.) — Variable speed drives up to 20000 h.p. eddy current couplings up to 800 h.p. brushless alternators.

Jeumont - Schneider (France) — Railway traction substation rectifiers, Electrolytic rectifier plant, AC to DC High power rectifiers.

Filotex (France) — Special cables, co-axial, broadcast, compensating, fire resistant.

Sprague — Capacitors of all types, transistors, integrated circuits, T.V. components.

Welwyn Electric (U.K.) — Resistors, potentiometers, Hybrid circuits.

Thomson — CSF (France) — Electronic Components, electronic systems, Broadcast equipment.

AT & AE

RCA 1800

MICROPROCESSOR

Binary Arithmetic Subroutines for
RCA COSMAC Microprocessors

MPM-206

**Binary Arithmetic Subroutines
for
RCA COSMAC Microprocessors**

Copyright 1976 by RCA Corporation
(All rights reserved under Pan-American Copyright Convention)

Information furnished by RCA is believed to be accurate and reliable. However, no responsibility is assumed by RCA for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of RCA.

Trademark(s) Registered ®
Marca(s) Registrada(s)

Foreword

The RCA CDP1801 and CDP1802 COSMAC Microprocessors are CMOS 8-bit register-oriented central processing units. They are suitable for use in a wide range of stored-program computer systems and products. Often, their applications require extended precision arithmetic calculations with possible interfacing of the system to BCD-oriented peripheral hardware. For such applications, this Manual provides a set of 16-bit 2's complement arithmetic subroutines designed to be operated on the RCA COSMAC Microprocessors. Versions are supplied for use with either the RCA CDP1801 or CDP1802. A suitable selection from this set of subroutines may be made for the calculations required in a specific application.

The subroutines described in this Manual are also available in source language on a floppy diskette CDP18S826 for use with the RCA Floppy Disk System CDP18S800.

In this Manual, the 29 Binary Arithmetic Subroutines are described, first in general and then in detail. A Standard Call and Return Technique for facilitating their use is provided along with complete listings. Additionally, BCD-to-binary and binary-to-BCD conversions are described. As a tutorial aid, a sample program is also provided.

Users requiring additional information on the available hardware, firmware, and software support systems should refer to the COSMAC technical literature. Reference should be made to the device technical data sheets and to the following publications:

- | | |
|---------|---|
| MPM-201 | User Manual for the RCA CDP1802 COSMAC Microprocessor |
| MPM-202 | Timesharing Manual for the RCA CDP1802 COSMAC Microprocessor |
| MPM-203 | Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor |
| MPM-208 | Operator Manual for the RCA COSMAC Development System |
| MPM-101 | User Manual for the CDP1801 COSMAC Microprocessor |
| MPM-102 | Program Development Guide for the CDP1801 COSMAC Microprocessor (Timesharing) |

Table of Contents

	Page
Foreword	3
Binary Arithmetic Subroutines	5
Introduction	5
Application	6
Basic Conventions	6
Subroutine Categories	7
Calling Sequence	8
Algorithms	8
Meaning of DF	8
Utility Routines	8
Defined Options	9
Timing	9
Detailed Description of Subroutines	11
Index by Category	11
Index — Alphabetically	11
Appendix A - Standard Subroutine Call and Return Technique	25
Appendix B (I) - Arithmetic Subroutine Listing for CDP1801	27
Appendix B (II) - Arithmetic Subroutine Listing for CDP1802	45
Appendix C - Sample Program	61

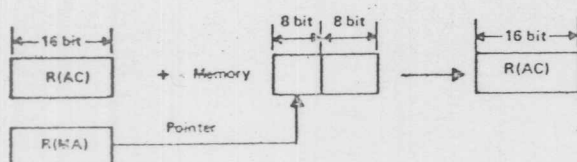
Binary Arithmetic Subroutines

Introduction

The Binary Arithmetic Subroutine Package given in this Manual is a set of 16-bit 2's complement arithmetic subroutines designed to be operated on a COSMAC microprocessor. The subroutines are coded in Level-I assembly language and require 1K-byte of memory space. These subroutines do not alter themselves and can be stored in Read-Only Memory. Different versions are available for the CDP1801 and CDP1802. The subroutines may be used individually or in any combination.

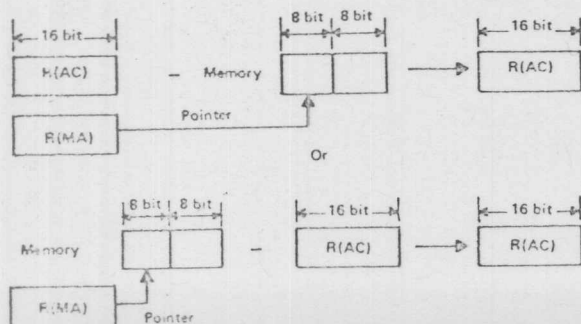
Four arithmetic functions are included:

1. 16-bit 2's complement Addition:



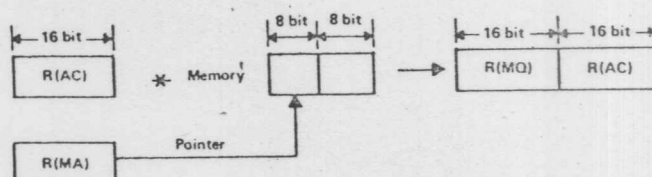
92CS-28092

2. 16-bit 2's complement Subtraction:



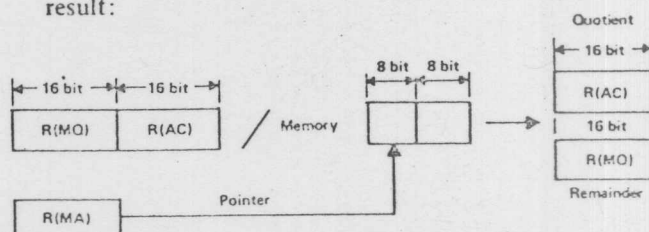
92CS-28089

3. 16-bit 2's complement Multiplication which yields 32-bit result:



92CS-28090

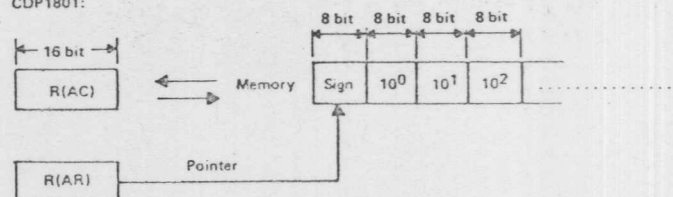
4. 32-bit 2's complement Division which yields 16-bit result:



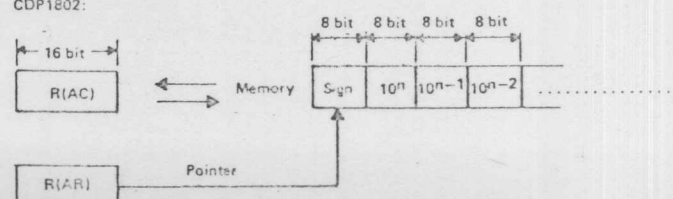
92CS-28088

BCD-to-binary and binary-to-BCD conversions together with utility routines (which can be used to save or restore the contents of registers) are provided.

CDP1801:



CDP1802:



92CS-28091

Applications

Various applications of these subroutines are possible. For example, a user may enter BCD numbers from external devices, manipulate these numbers in 2's complement arithmetic functions, and finally output the result in BCD form back to external devices. A diagram of such an application is given in Fig. 1.

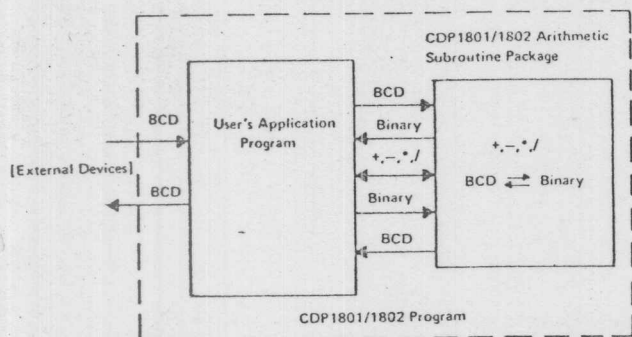


Fig. 1 — Example of microprocessor system using the arithmetic subroutine package.

It is possible to use the arithmetic function(s) only, as diagrammed in Fig. 2, or a combination of these subroutines to suit the user's needs.

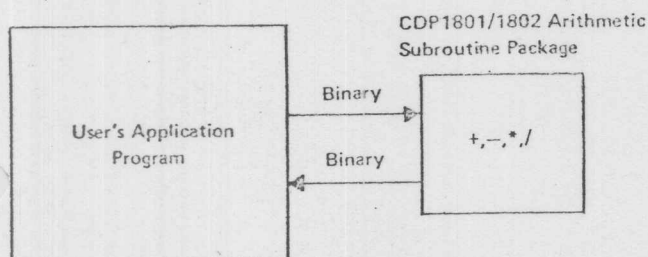


Fig. 2 — Application using arithmetic functions only.

Basic Conventions

All of the subroutines in this package follow the call and return conventions described in the User Manual for the RCA CDP1802 COSMAC Microprocessor (MPM-201). A stack area in RAM is also implied. Symbolic register names are used to permit custom modifications and register allocation. The standard registers' definitions and their functions are:

R(SP) = R2	.. The stack pointer.
R(PC) = R3	.. The program counter used by all the subroutines.
R(CALL) = R4	.. Pointer to CALL routine.
R(RETN) = R5	.. Pointer to RETURN routine.
R(LINK) = R6	.. Pointer to arguments passed from main program.
R(AR) = RA	.. Pointer to BCD digits in memory. Used in BCD conversion routines.
R(NR) = RB	.. Counter for the number of BCD digits. Used in BCD conversion routines.
R(CR) = RC	.. Scratchpad and counter. Used in multiply, divide and others.
R(MA) = RD	.. Pointer to the address of one of the two operands in add, subtract, multiply and divide operations.
R(MQ) = RE	.. 16-bit accumulator or sign extension. Used in multiplication and division to hold the most significant half of the product and dividend. It is also used to store the remainder after a divide operation.
R(AC) = RF	.. 16-bit accumulator. Used as one of the two operands for all computations.

Subroutine Categories

A list of the binary arithmetic subroutines by category together with a description of their functions follows.

The secondary subroutines shown are used internally by the other subroutines only. Their functions, therefore,

will not be described in further detail. For the interested user, commented coding is provided on the actual listing of the binary arithmetic subroutine package. These secondary routines are relatively short and self-explanatory.

I. ADDITION

NAME	ARGUMENT	FUNCTION
ADDOP	aaaa	$R(AC) + M(\text{OPERAND POINTER}) \rightarrow R(AC)$
ADD	---	$R(AC) + \text{OPERAND} \rightarrow R(AC)$
ADDCON	nnnn	$R(AC) + \text{CONSTANT} \rightarrow R(AC)$
ADDST	---	$R(AC) + \text{two bytes on stack} \rightarrow R(AC)$

II. SUBTRACTION

SDOP	aaaa	$M(\text{OPERAND POINTER}) - R(AC) \rightarrow R(AC)$
SD	---	$\text{OPERAND} - R(AC) \rightarrow R(AC)$
SDCON	nnnn	$\text{CONSTANT} - R(AC) \rightarrow R(AC)$
SMOP	aaaa	$R(AC) - M(\text{OPERAND POINTER}) \rightarrow R(AC)$
SM	---	$R(AC) - \text{OPERAND} \rightarrow R(AC)$

III. MULTIPLICATION

MPYOP	aaaa	$R(AC) * M(\text{OPERAND POINTER}) \rightarrow [R(MQ); R(AC)]$
MPY	---	$R(AC) * \text{OPERAND} \rightarrow [R(MQ); R(AC)]$

IV. DIVISION

DIVOP	aaaa	$[R(MQ); R(AC)] / M(\text{OPERAND POINTER}) \rightarrow R(AC)$
DIV0	---	zero divide check and sign extension
DIV	---	quotient overflow check
DIVQ	---	$[R(MQ); R(AC)] / \text{OPERAND} \rightarrow R(AC)$

V. BINARY-CODED-DECIMAL NUMBER CONVERSION

CBD	aaaa.nn	Binary in $R(AC) \rightarrow \text{BCD}$
CDB	aaaa.nn	$\text{BCD} \rightarrow \text{Binary in } R(AC)$

VI. UTILITY ROUTINES

PUSHAC	---	$R(AC) \rightarrow \text{stack}$
PUSHCQ	---	$R(MQ), R(MA), R(CR) \rightarrow \text{stack}$
POPAC	---	$\text{stack} \rightarrow R(AC)$
POP	---	$\text{stack} \rightarrow \text{discard}$
POPCQ	---	$\text{stack} \rightarrow R(MQ), R(MA), R(CR)$
LOADOP	aaaa	$M(\text{OPERAND POINTER}) \rightarrow R(AC)$
LOAD	---	$\text{OPERAND} \rightarrow R(AC)$
LODCON	nnnn	$\text{CONSTANT} \rightarrow R(AC)$
STOROP	aaaa	$R(AC) \rightarrow M(\text{OPERAND POINTER})$
STORE	---	$R(AC) \rightarrow \text{OPERAND}$
COMPOP	aaaa	If $R(AC) \geq M(\text{OPERAND POINTER})$ then $I \rightarrow \text{DF}$
COMP	---	If $R(AC) \geq \text{OPERAND}$ then $I \rightarrow \text{DF}$
TEST	---	compare $R(AC)$ with 0
SWAPAQ	---	$R(AC) \rightleftharpoons R(MQ)$

VII. SECONDARY ROUTINES

DABS*		$\text{ABS}[R(MQ)] \quad \text{ABS}(\text{OPERAND}) \rightarrow R(MQ)$
DSM*		entry point of DABS
DVA*		entry point of DABS
DSHL*		$\text{MQ} * 2 \rightarrow \text{MQ}$
DZS		clear memory locations for BCD

NOTES:

aaaa is two bytes address of operand. * found in CDP1801 version only.
nnnn is two bytes constant literal.

Calling Sequence

The calling sequences of the subroutines are similar; the only differences are the nature of their arguments.

Some routines deal implicitly with an operand on the stack, or with an operand pointed to by the address register. Some fetch the address of the operand from the call sequence argument list, and others fetch the actual argument from the call sequence. Typically, a routine which fetches the address of an operand is used if the operand is a variable. On the other hand, a routine that fetches the actual argument is used if the operand is a constant. For instance, if the function $A=B/C+5$ is to be performed, the calling sequence of the divide and add routines could be:

SEP CALL

, A(DIVOP) .. assume [R(MQ);R(AC)] contain the variable B.
 .. address of the variable C in memory.
 .. now R(AC) contains the quotient of the
 .. operation B/C.

SEP CALL

, A(ADDCON)
 , X'0005' .. the constant 5 is to be added to the
 .. quotient of B/C stored in R(AC) previously.
 .. now R(AC) contains B/C+5

Algorithms

The add and subtract routines use ordinary 2's complement add and subtract methods. For multiplication, the add-shift algorithm is implemented. The divide routine uses the nonrestoring method. These algorithms can be found in computer arithmetic books. For example, see *Digital Computer Design Fundamentals* by Yaohan Chu, McGraw-Hill Book Company.

The BCD-to-binary conversion is accomplished by taking the digits in order, starting with the most significant, and adding them to ten times the previous partial result—successively.

The binary-to-BCD conversion for the CDP1801 is implemented in the reverse order. The binary number in register AC is divided by ten successively until R(AC) becomes zero. On each iteration, the remainder is the corresponding BCD digit starting from the least significant. DZS is an internal subroutine used to initialize all the digits to zero.

The implementation of the binary-to-BCD conversion for the CDP1802 is faster than that for the CDP1801. It is described in the book referenced above.

Meaning of DF

The add, subtract, divide, BCD-to-binary conversion, and binary-to-BCD conversion routines affect DF. The DF is set if the result has overflowed. Since no overflow condition is possible for multiplication, the multiply routine sets DF if the product is greater than 16 bits. Some of the subroutines also use DF to indicate the terminal condition of an operation. For example, the compare subroutine sets DF if the content of R(AC) is greater than or equal to the operand pointed to by register MA. Thus, it is assumed and is essential that the subroutine linkage routines do not alter the state of DF. The "Standard Subroutine Call and Return Technique" is assumed. Appendix A contains a listing typical of these programs.

Utility Routines

Four of the general registers R(AC), R(MQ), R(MA), and R(CR), are used by various routines. For programs that need these registers, special routines have been included which save them on the stack, and later restore them. Accordingly, the register save and restore routines are completely independent. In particular, these routines are partitioned into two sets: routines to push and pop the accumulator, and routines to push and pop the other three registers as a block. The separate routines for the accumulator permit intermediate results to be saved on the stack for later use in computation. In addition, a routine is provided to swap or exchange the accumulator with the accumulator extension, which is used in multiplication and division.

It is assumed that the stack pointer normally points to one byte above the first of two bytes stored by the subroutine call operation (i.e., the last pushed, having the lower memory address), and that these two bytes are normally popped off by the return operation. Note that this assumption is consistent with the standard subroutine linkage conventions. Thus, to save data on the stack, the data must be inserted beneath those first two bytes, so that when they are popped off, the saved data will be on top. Similarly, when the data is to be recovered, it must be fetched from beneath the top two bytes, then the gap closed up. One of the entries to the binary add routine also permits using the virtual top of the stack as one operand (that is, the top of the stack as seen by the calling program).

Defined Options

Some of the lines in the listing have one or more flags on the right margin. They consist of a symbol enclosed within parentheses. These symbols are used to flag "Defined Options" which are specified in the notes to follow. Usually to invoke a particular option, all of the lines flagged by the same character are altered in some specified way.

(@) Separate Stack

While it is expected that the user will utilize the standard subroutine linkage, there is no requirement that the stack pointer used for the binary arithmetic be the same as that for the subroutine linkage. Since some of these routines do not themselves make use of the call linkages, it is not always necessary to use a stack for the

lls. If a stack is not needed, subroutines PUSHCQ, PUSHAC, POPCQ and POPAC are inappropriate. If their functions are desired, the routines must be rewritten, omitting the operation of slicing off the top of the stack and working under it (actually this results in the routines being trivially simple, so the user should have no difficulty recoding them). If the arithmetic from the top of the stack is still a desirable feature, the lines flagged by (@) must be deleted, so that R(MA) will point to the actual top of the stack.

(0) 16-Bit Dividend

By retaining the lines so flagged, and deleting the lines flagged (/), the divide routine will assume that all dividends fit entirely into the R(AC), and that error conditions can occur only if the divisor is zero.

(/) 32-Bit Dividend

By retaining the lines so flagged, and deleting the lines flagged (0), the divide routine will assume that all dividends occupy the full 32 bits of the R(AC) and R(MQ), and that a divide fault is possible only if the quotient re-

ult of the dividend divided by the divisor would exceed the 16 bits of the R(AC), or if the divisor is zero (which is equivalent to an infinite quotient).

(0) (/) No Divide Check

By deleting all the lines flagged by either (0) or (/), the divide routine will perform no testing on the possibility of a divide fault. This deletion poses the simple hazard that the numbers resulting from the operation will be meaningless if a divide fault occurs, and no indication of the error will be returned to the calling routine. Note that in this option, the dividend is assumed to be 32 bits.

If the user needs to make alterations to the routines, or if some are omitted, their placement in memory may need changing to eliminate possible branches out of page.

Timing

Timing measurements, given in Fig. 3, were taken on the arithmetic and code conversion subroutines. For the CDP1801 version, the speed is limited by the size of the program since the objective is to fit the total package into 1-K of memory. However, with the CDP1802 instruction set, it is possible to meet this memory space requirement while optimizing for speed. All the measurements were based on a clock rate of 2 MHz. The actual timing can be rescaled depending on the user's application system clock rate. For instance, if the CDP1802 is operated at a 6-MHz clock rate, the time shown is reduced by a factor of three.

I. Arithmetic Functions

	ADD	SUBTRACT	MULTIPLY	DIVIDE
Best:	0.152 ms	0.168 ms	3.112 ms	13.80 ms
Worst:	0.264 ms	0.32 ms	9.782 ms	16.15 ms

II. BCD $\rightleftharpoons$ BINARY

	BINARY - BCD	BCD - BINARY
Best:	15.36 ms	5.84 ms
Worst:	73.48 ms	29.76 ms

Fig. 3(a)—Timing measurements for the RCA CDP1801 COSMAC Microprocessor (based on 2-MHz clock rate).

I. Arithmetic Functions

	ADD	SUBTRACT	MULTIPLY	DIVIDE
Best:	0.136 ms	0.128 ms	2.792 ms	4.496 ms
Worst:	0.224 ms	0.256 ms	4.223 ms	5.856 ms

II. BCD $\rightleftharpoons$ BINARY

	BINARY - BCD	BCD - BINARY
Best:	4.37 ms	0.31 ms
Worst:	9.27 ms	2.67 ms

Fig. 3(b)—Timing measurements for the RCA CDP1802 COSMAC Microprocessor (based on 2-MHz clock rate).

Detailed Description of Subroutines

The following material provides a detailed description of each subroutine and how it is used. For ease of use this information is given in a standard format which includes the subroutine identification, its function, the calling

procedure, the registers used, other subroutines used, the meaning of DF on return, the number magnitude, the time range for the subroutine, its length in bytes, and, where helpful, an example.

Index by Category

Subroutine	Page
I. ADDITION	
ADDOP	12
ADD	12
ADDCON	12
ADDST	13
II. SUBTRACTION	
SDOP	13
SD	13
SDCON	14
SMOP	14
SM	14
III. MULTIPLICATION	
MPYOP	15
MPY	15
IV. DIVISION	
DIVOP	16
DIVO	16
DIV	16
DIVQ	16
V. BINARY-CODED-DECIMAL NUMBER CONVERSION	
CBD	18
CDB	19
VI. UTILITY ROUTINES	
PUSHAC	20
PUSHCQ	20
POPAC	20
POP	20
POPCQ	20
LOADOP	21
LOAD	21
LODCON	21
STOROP	22
STORE	22
COMPOP	22
TEST	23
SWAPAQ	23

Index - Alphabetically

Subroutine	Page
ADD	12
ADDCON	12
ADDOP	12
ADDST	13
CBD	18
CDB	19
COMP	22
COMPOP	22
DIV	16
DIVO	16
DIVOP	16
DIVQ	16
LOAD	21
LOADOP	21
LODCON	21
MPY	15
MPYOP	15
POP	20
POPAC	20
POPCQ	20
PUSHAC	20
PUSHCQ	20
SD	13
SDCON	14
SDOP	13
SM	14
SMOP	14
STORE	22
STOROP	22
SWAPAQ	23
TEST	23

1. Subroutine: **ADDOP**
ADD
2. Function:
ADDOP and ADD are two entries of a subroutine that add the content of R(AC) and a two-byte operand in memory pointed to by R(MA). The sum is stored in R(AC).
ADDOP fetches the address of operand from the calling program and stores it in R(MA).
ADD assumes the address of operand is already in R(MA).
3. Calling Procedure:
 .. enter from ADDOP
SEP CALL
 A(ADDOP)
 A(OPR) .. where OPR is
 .. the address of the
 .. operand.
 .. enter from ADD
SEP CALL
 A(ADD) .. R(MA) contains the address of
 .. operand.
4. Registers Used:
R(LINK) for ADDOP only, R(MA), R(AC), R(SP).
5. Other Subroutines Used:
ADDOP : should be followed in memory by ADD.
ADD : none
6. On Return:
DF = 0 means addition was successful.
DF = 1 means the sum has exceeded the maximum or minimum range of representable numbers.
7. Number:
Representation: Signed 2's complement
Width: 16 bits
Range: $8000 \leq \text{number}_{16} \leq 7FFF$
 $-32768 \leq \text{number}_{10} \leq 32767$
8. Time:
CLOCK RATE: 2 MHz 6.4 MHz
CDP1801 Best 0.152 — ms
Worst 0.264 — ms
CDP1802 Best 0.136 0.042 ms
Worst 0.192 0.060 ms
9. Length:
CDP1801 CDP1802
version version
45 30 bytes
ADDOP 41 26 bytes
ADD 37 22 bytes
10. Example:
 .. assume AC = FFFE (= -2);
 .. assume M(OPR) = #0001
 .. enter from ADDOP
SEP CALL
 A(ADDOP)
 A(OPR) .. address of operand
[code] .. R(AC) is now FFFF (= -1)
 .. with DF = 0

1. Subroutine: **ADDCON**
2. Function:
This subroutine adds the content of R(AC) to the two-byte constant passed from the calling program. The sum is placed in R(AC).
3. Calling Procedure:
SEP CALL
 A(ADDCON)
 ,X'nnnn' .. where nnnn is the
 .. constant to be added to R(AC).
4. Registers Used:
R(LINK), R(MA), R(AC), R(SP).
5. Other Subroutine Used:
ADD
6. On Return:
DF = 0 means addition was successful.
DF = 1 means the sum exceeded the maximum or minimum range of representable numbers.
7. Number:
Representation: Signed 2's complement
Width: 16 bits
Range: $8000 \leq \text{number}_{16} \leq 7FFF$
 $-32768 \leq \text{number}_{10} \leq 32767$
8. Time:
CLOCK RATE: 2 MHz 6.4 MHz
CDP1801 Best 0.208 — ms
Worst 0.288 — ms
CDP1802 Best 0.192 0.060 ms
Worst 0.216 0.067 ms
9. Length:
CDP1801 CDP1802
version version
45 30 bytes

1. Subroutine: ADDST

2. Function:

ADDST adds the content of R(AC) to the top two bytes of stack. These two bytes should be stored in a sequence such that the more significant part is on the top and the less significant part is at the bottom.

3. Calling Procedure:

SEP CALL

,A(ADDST) .. 2-byte constant is stored .. on top of stack.

4. Registers Used:

R(LINK), R(MA), R(AC), R(SP).

5. Other Subroutine Used:

ADD

6. On Return:

DF = 0 means addition was successful.

DF = 1 means the sum exceeded the maximum or minimum range of representable numbers.

7. Number:

Representation: Signed 2's complement

Width: 16 bits

Range: $8000 \leq \text{number}_{16} \leq 7FFF$ or $-32768 \leq \text{number}_{10} \leq 32767$

8. Time:

CLOCK RATE: 2 MHz 6.4 MHz

	Best	0.218	—	ms
CDP1801	Worst	0.298	—	ms

	Best	0.204	0.064	ms
CDP1802	Worst	0.224	0.070	ms

9. Length:

	CDP1801 version	CDP1802 version	bytes
	46	31	

1. Subroutine: SDOP

SDOP
SD

2. Function:

SDOP and SD are two different entries of a subroutine that subtract R(AC) from the contents of two bytes in memory that is pointed to by R(MA).

SDOP fetches the address of minuend from the calling program and stores it in R(MA).

SD assumes the address of minuend is already in R(MA).

3. Calling Procedure:

.. to enter from SDOP

SEP CALL

,A(SDOP)

,A(MINU) .. where MINU is the address .. of the minuend.

.. to enter from SD

SEP CALL

,A(SD) .. the address of minuend should be .. in R(MA).

4. Registers Used:

R(LINK) for SDOP only, R(MA), R(AC), R(SP).

5. Other Subroutines Used:

SDOP: should be followed in memory by SD.

SD : none

6. On Return:

DF = 0 means subtraction was successful.

DF = 1 means subtraction was unsuccessful because the difference has exceeded the maximum or minimum range of representable numbers.

7. Number:

Representation: Signed 2's complement

Width: 16 bits

Range: $8000 \leq \text{number}_{16} \leq 7FFF$ or $-32768 \leq \text{number}_{10} \leq 32767$.

8. Time:

CLOCK RATE: 2 MHz 6.4 MHz

	Best	0.168	—	ms
CDP1801	Worst	0.256	—	ms

	Best	0.128	0.040	ms
CDP1802	Worst	0.184	0.057	ms

9. Length:

	CDP1801 version	CDP1802 version	bytes
SDOP	35	24	
SD	31	20	

10. Example:

[code] .. assume R(AC) = X'000A'
.. assume M(MINU) = X'0005'

SEP CALL

,A(SDOP) .. enter from SDOP

,A(MINU) .. address of minuend will be .. fetched by SDOP

[code] .. R(AC) is now FFFB (= -0005)
.. with DF = 0.

1. Subroutine: SDCON
2. Function:
This subroutine subtracts AC from a 16-bit constant and stores the difference in AC.
3. Calling Procedure:
SEP CALL
A(SDCON)
'X'nnnn' .. where nnnn is the constant.
4. Registers Used:
R(LINK), R(MA), R(AC), R(SP).
5. Other Subroutine Used:
SD
6. On Return:
DF = 0 means subtraction was successful.
DF = 1 means subtraction was unsuccessful because the difference exceeded the maximum or minimum range of representable numbers.
7. Number:
Representation: Signed 2's complement
Width: 16 bits
Range: $8000 \leq \text{number}_{16} \leq 7FFF$ or $-32768 \leq \text{number}_{10} \leq 32767$
8. Time:
- | | | 2 MHz | 6.4 MHz | |
|---------|-------|-------|---------|----|
| CDP1801 | Best | 0.216 | — | ms |
| | Worst | 0.272 | — | ms |
| CDP1802 | Best | 0.184 | 0.057 | ms |
| | Worst | 0.208 | 0.065 | ms |
9. Length:
- | | CDP1801
version | CDP1802
version | |
|--|--------------------|--------------------|-------|
| | 39 | 28 | bytes |
10. Example:
[code] .. assume R(AC) = X'0002'
SEP CALL
A(SDCON)
'X'0002' .. constant is X'0005'
[code] .. R(AC) is now X'0003' with DF = 0

1. Subroutine: SMOP
SM
2. Function:
SMOP and SM are two different entries of a subroutine that subtract the contents of two bytes in memory, which is pointed to by R(MA), from R(AC). SMOP fetches the address of subtrahend from the calling program and stores it in R(MA). SM assumes the address of subtrahend is already in R(MA).
3. Calling Procedure:
.. to enter from SMOP
SEP CALL
A(SMOP)
A(SUBT) .. where SUBT is the address of the subtrahend.
.. to enter from SM
SEP CALL
A(SM) .. R(MA) should contain the address of subtrahend.
4. Registers Used:
R(LINK) for SMOP only, R(MA), R(AC), R(SP).
5. Other Subroutines Used:
SMOP: should be followed in memory by SM.
SM : none
6. On Return:
DF = 0 means subtraction was successful.
DF = 1 means subtraction was unsuccessful because the difference has exceeded the maximum or minimum range of representable numbers.
7. Number:
Representation: Signed 2's complement
Width: 16 bits
Range: $8000 \leq \text{number}_{16} \leq 7FFF$ or $-32768 \leq \text{number}_{10} \leq 32767$
8. Time:
- | | | 2 MHz | 6.4 MHz | |
|---------|-------|-------|---------|----|
| CDP1801 | Best | 0.216 | — | ms |
| | Worst | 0.320 | — | ms |
| CDP1802 | Best | 0.192 | 0.060 | ms |
| | Worst | 0.256 | 0.080 | ms |
9. Length:
- | | CDP1801
version | CDP1802
version | |
|------|--------------------|--------------------|-------|
| SMOP | 49 | 34 | bytes |
| SM | 45 | 30 | bytes |
10. Example:
[code] .. assume M(100; 101) = 0001 and
.. R(MA) = 0100 and R(AC) = 8000
SEP CALL
A(SM) .. take SM entry
[code] .. DF = 1
.. difference exceeded the
.. maximum range of negative
.. number

1. Subroutine: **MPYOP**
MPY
2. Function:
MPYOP and MPY are two different entries of a subroutine that multiply R(AC) by the contents of two bytes in memory which are pointed to by R(MA). The product is placed in R(MQ) and R(AC) with the highest bit of R(MQ) as the sign bit and the lowest bit of R(AC) as the least significant bit.
- MPYOP fetches the address of multiplier from the calling program and stores it in R(MA).
- MPY assumes the address of multiplier is already in R(MA).
3. Calling Procedure:
- SEP CALL .. to enter from MPYOP
,A(MPYOP)
,A(MPLR) .. where MPLR is the address
 .. of the multiplier.
- SEP CALL .. to enter from MPY
,A(MPY) .. the address of multiplier
 .. should be in R(MA).
4. Registers Used:
R(LINK) for MPYOP only, R(MA), R(MQ), R(AC), R(CR).
5. Other Subroutines Used:
MPYOP: should be followed in memory by MPY.
MPY : DSM, DVA (for CDP1801 version only).
6. On Return:
DF = 0 means product ≤ 16 bits
DF = 1 means product > 16 bits

7. Number:
Representation: Signed 2's complement
Width: Multiplicand and Multiplier:
 16 bits
Product:
 32 bits
Range: Multiplicand and Multiplier:
 $8000 \leq \text{number}_{16} \leq 7FFF$ or
 $-32768 \leq \text{number}_{10} \leq 32767$
Product:
 $C0008000 \leq \text{number}_{16} \leq 40000000$
 $-1073709056 \leq \text{number}_{10} \leq 1073741824$

8. Time:
- | CLOCK RATE | 2 MHz | 6.4 MHz | |
|---------------|-------|---------|----|
| CDP1801 Best | 3.112 | — | ms |
| CDP1801 Worst | 9.782 | — | ms |
| CDP1802 Best | 2.792 | 0.872 | ms |
| CDP1802 Worst | 4.223 | 1.320 | ms |

9. Length:
- | | CDP1801
version* | CDP1802
version | |
|-------|---------------------|--------------------|-------|
| MPYOP | 131 | 81 | bytes |
| MPY | 127 | 77 | bytes |

*Including secondary subroutines.

10. Example:
- [code] .. assume M(MPLR) = X'0001'
 .. assume R(AC) = FFFF(= -0001)
- SEP CALL
,A(MPYOP) .. enter from MPYOP
,A(MPLR)
[code] .. [R(MQ); R(AC)] = FFFFFFFF(= -1)
 .. with DF = 0 since the magnitude
 .. of product is less than 15 bits

1. Subroutine:

DIVOP
DIV0
DIV
DIVQ

Alternatively, the code could fall through DIV and DIVQ if the option of quotient overflow check is selected.

2. Function:

DIVOP, DIV0, DIV, DIVQ are four different entries of a subroutine which divide the contents of [R(MQ); R(AC)] by the contents of two bytes in memory pointed to by R(MA).

Upon returning from the subroutine, R(AC) contains the quotient and R(MQ) contains the remainder.

DIVOP fetches the address of divisor from the calling program and stores it in R(MA).

DIV0 assumes the address of divisor is already in R(MA) and the magnitude of dividend does not exceed 15 bits. It extends the sign bit of R(AC) into R(MQ), and does a divide check (divisor = 0).

DIV also assumes the address of divisor is already in R(MA). It checks dividend against divisor to make sure that the magnitude of quotient will not exceed 15 bits. The divisible numbers are shown on the following page.

DIVQ assumes R(MA) contains the address of divisor. It divides [R(MQ); R(AC)] by the operand in memory which is pointed to by R(MA). When returning to the calling program, R(AC) contains the quotient with remainder stored in R(MQ). This entry does not check condition; hence, it is the user's responsibility to determine if the quotient is meaningful.

3. Calling Procedure:

SEP CALL .. enter from DIVOP
A(DIVOP)
A(DIVR) .. where DIVR is the
 .. address of divisor.
 .. enter from the other entry points
SEP CALL
A(DIV0) .. or DIV, or DIVQ
 .. the address of divisor
 .. should be in MA.

4. Registers Used:

DIVOP: R(LINK), R(MA).
DIV0: R(MA), R(AC), R(MQ).
DIV: R(CR), R(MQ), R(AC), R(MA).
DIVQ: R(CR), R(MQ), R(AC), R(MA), R(SP).

5. Other Subroutines Used:

DIVOP: The code should fall through DIV0 and DIVQ if zero divide check and sign extension option (which is performed by DIV0) is chosen.

DIV0: The code should fall through DIV, DIVQ.

DIV: (1) The code should fall through DIVQ.
(2) DSHL, DABS (for CDP1801 only).

DIVQ: DSHL, DABS (for CDP1801 only).

6. On Return:

DF = 0 means division was successful.

DF = 1 means:

- (1) If subroutines are in sequence of DIVOP, DIV0, DIVQ, or DIV0, DIVQ: Division was unsuccessful because [R(MQ); R(AC)] division by 0 was attempted.
- (2) If subroutines are in sequence of DIVOP, DIV, DIVQ or DIV, DIVQ: Division was unsuccessful because the magnitude of dividend is too large and, consequently, R(AC) will overflow.

The contents of R(MQ) and R(AC) are not changed if DF is set to 1.

7. Number:

Representation: Signed 2's complement

Width: Dividend: 32 bits

Divisor, Quotient, Remainder: 16 bits

Range: Dividend (absolute):

$C0000001 \leq \text{number}_{16} \leq 400007FFF$

$-1073741823 \leq \text{number}_{10} \leq 1073774591$

Divisor, Remainder, Quotient:

$8000 \leq \text{number}_{16} \leq 7FFF$

$-32768 \leq \text{number}_{10} \leq 32767$

8. Time:

CLOCK RATE:		2 MHz	6.4 MHz	
CDP1801	Best	13.38	—	ms
	Worst	15.76	—	ms
CDP1802	Best	4.496	1.405	ms
	Worst	5.856	1.830	ms

9. Length:

The numbers indicated here are the lengths of each entry of the divide subroutine. Thus, the actual length of the subroutine should be the sum of the entries that are used.

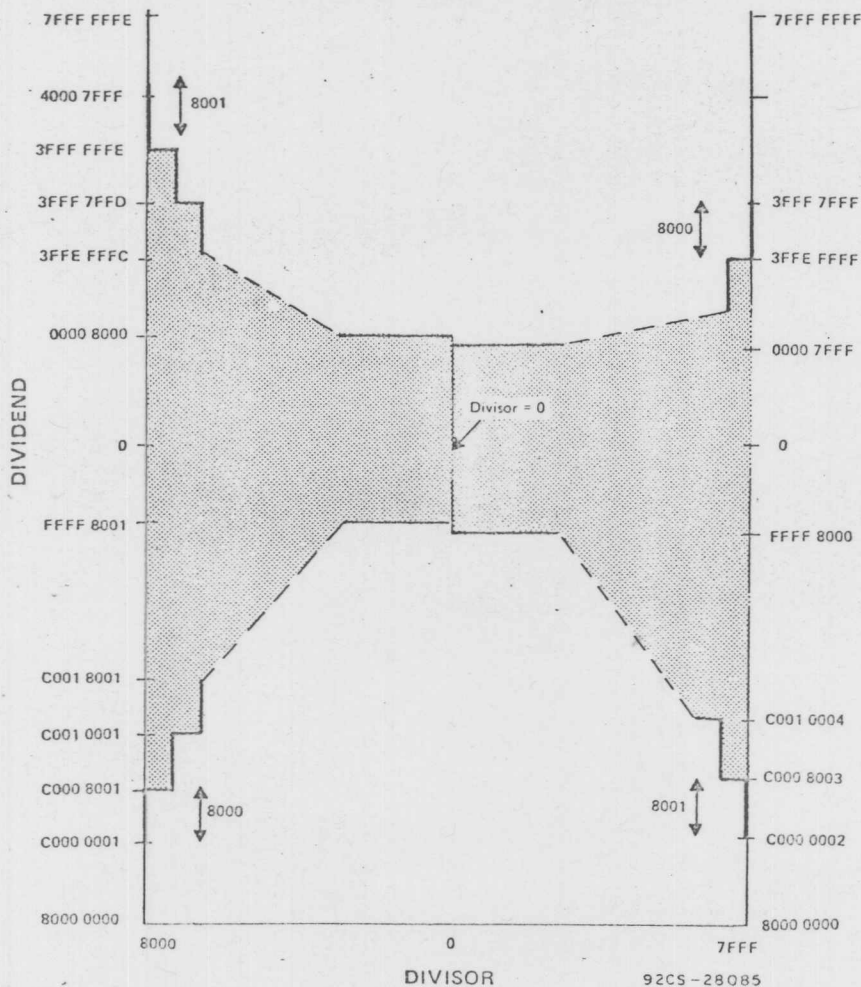
	CDP1801 version	CDP1802 version	
DIVOP	4	4	bytes
DIV0	20	18	bytes
DIV	170	168	bytes
DIVQ	173	114	bytes

10. Example 1:

[code] .. assume M(DIVR) = X'0003'
 .. assume R(AC) = X'000A'
 .. enter DIVOP, DIV0, DIVQ
 SEP CALL
 ,A(DIVOP) .. followed by DIV0, DIVQ
 ,A(DIVR) .. address of divisor
 [code] .. [R(MQ); R(AC)] is now 00010003
 .. where remainder = 0001
 .. and quotient = 0003
 .. with DF = 0

Example 2:

[code] .. assume M(DVSR) = X'0002'
 [code] .. and [R(MQ); R(AC)] = 00500001
 .. enter DIVOP, DIV, DIVQ
 SEP CALL
 ,A(DIVOP) .. followed by DIV, DIVQ
 ,A(DVSR) .. address of divisor
 [code] .. DF = 1 since [R(MQ); R(AC)]
 .. (= 00500001) divided by
 .. 0002 would result in
 .. overflow of AC (Quotient
 .. is too large to be able to
 .. store in 16 bits)



Shaded area indicates divisible numbers.

1. Subroutine:

CBD

2. Function:

CBD converts the signed 2's complement number in R(AC) to binary-coded-decimal (BCD) numbers. The address for the storing is a two-byte hex number passed from the calling program. The number of digits (including sign) of this BCD number is a one-byte hexadecimal number passed also from the calling program to CBD. On return, the BCD result is stored in a sequence such that the sign of the BCD numbers is followed for the CDP1801 version by the least significant digit, followed by the second least significant digit, and for the CDP1802 version, by the most significant digit, followed by the second most significant digit, etc.

3. Calling Procedure:

SEP CALL

,A(CBD)

,A(BCD)

#nn

.. address for BCD result
 .. where nn is a one-byte
 .. constant in hex to indicate
 .. the maximum number of
 .. digits for result (including
 .. sign).

4. Registers Used:

R(LINK), R(AR), R(NR), R(AC), R(MA).

5. Other Subroutines Used:

SDCON

DZS

PUSHAC

DIVQ

For CDP1801 version only.

6. On Return:

DF = 0 means conversion was successful.

DF = 1 means the number of digits of BCD (and sign) is larger than the length argument passed from the calling program. (Overflowed!)

7. Number:

Representation: Signed 2's complement R(AC).

Width: R(AC): 16 bits

BCD: $2 \leq \text{length} \leq 6$ Range: $8000 \leq \text{AC} \leq 7FFF$ $-32768 \leq \text{BCD} \leq 32767$

Sign of BCD: #0B = +

#0D = -

8. Time:

CLOCK RATE: 2 MHz 6.4 MHz

CDP1801	Best	15.36	—	ms
	Worst	73.48	—	ms

CDP1802	Best	4.37	1.366	ms
	Worst	9.27	2.897	ms

9. Length:

CDP1801 version*	CDP1802 version	bytes
339	83	

*Including secondary subroutines.

10. Example:

.. assume AC = FFE0 -32_{10}
 .. assume BCD is to be stored in memory
 .. starting at address A(RSLT).

SEP CALL

,A(CBD)

,A(RSLT) .. address of BCD to be stored

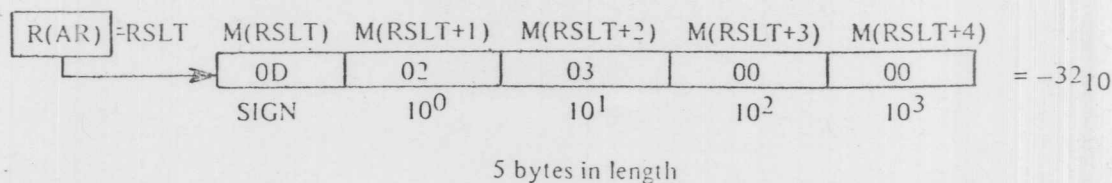
#05 .. only three digit locations will be used.

The last two locations will be
 cleared to zeros.

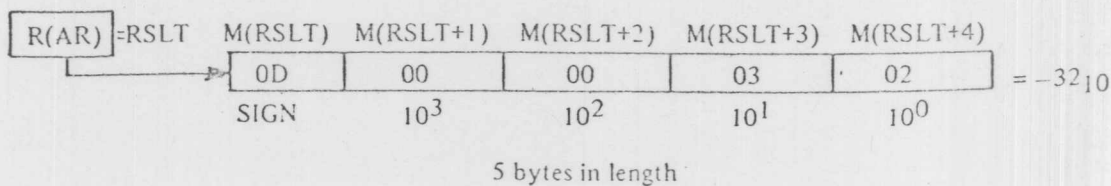
.. the BCD number will be stored in
 memory pointed to

.. by R(AR).

For the CDP1801 version, the above conversion will
 produce the following:



For the CDP1802 version, the above conversion will
 produce the following:



1. Subroutine: CDB
2. Function:
CDB converts the binary-coded-decimal (BCD) number stored in memory, whose address is passed from the calling program, into a signed 2's complement number and stores it in R(AC). The length of the BCD number, i.e., the number of digits and the sign, is a one-byte argument that is also passed from the calling program. The BCD number should be stored in memory in a sequence such that the first byte is the sign and, for the CDP1801, the second byte is the least significant digit, the third byte is the second least significant digit, etc.; for the CDP1802, the second byte is the most significant digit, the third byte is the second most significant digit, etc.

The sign of this BCD number is represented by #0B(+) and #0D(-).

3. Calling Procedure:
SEP CALL
A(CDB)
A(BCD number) .. address of the BCD number
#nn .. length of the BCD, includes .. sign.
4. Registers Used:
R(LINK), R(AR), R(NR), R(AC).

5. Other Subroutines Used:

PUSHAC
LODCON
STORE
ADDST
SDCON
MPY

For CDP1801 version only.

6. On Return:
DF = 0 means conversion was successful.
DF = 1 means conversion failed because the BCD number is either too large or too small to be representable as a signed 16-bit 2's complement number.
7. Number:
Representation: Signed 2's complement R(AC).
Width: R(AC): 16 bits
BCD: $2 \leq \text{length} \leq 6$
Range: $8000 \leq R(AC) \leq 7FFF$
 $-32768 \leq \text{BCD} \leq 32767$
8. Time:
CLOCK RATE: 2 MHz 6.4 MHz
CDP1801 Best 5.84 — ms
Worst 29.24 — ms
CDP1802 Best 0.31 0.097 ms
Worst 2.67 0.834 ms
9. Length:
CDP1801 version* 342
CDP1802 version 107 bytes

*Including secondary subroutines.

10. Example:
.. assume the BCD number, +1536, is to be
.. converted to signed 2's complement number
.. and the BCD is stored in the memory as

M(BCD)	M(BCD+1)	M(BCD+2)	M(BCD+3)	M(BCD+4)	
0B	X6	X3	X5	X1	= 1536 ₁₀
SIGN	10 ⁰	10 ¹	10 ²	10 ³	

for the CDP1801 version, or

M(BCD)	M(BCD+1)	M(BCD+2)	M(BCD+3)	M(BCD+4)	
0B	X1	X5	X3	X6	= 1536 ₁₀
SIGN	10 ³	10 ²	10 ¹	10 ⁰	

for the CDP1802 version.

where X is ignored by CDB, i.e., the BCD digits can be stored as ASCII numbers.

SEP CALL
A(CDB)
A(BCD) .. address of BCD
#05 .. 4 digits + sign = length of BCD
.. R(AC) now contains 0600₁₆(= 1536₁₀)

- 1. Subroutine:** PUSHAC
- 2. Function:**
This subroutine pushes the content of R(AC) onto stack with R(AC).1 on the top and R(AC).0 at the bottom. Upon returning, R(MA) is left pointing to R(AC).1 (one byte below the top of the stack, i.e., $R(MA) = R(SP)+1$).
- 3. Calling Procedure:**
SEP CALL
A(PUSHAC)
- 4. Registers Used:**
R(MA), R(AC), R(SP).
- 5. Other Subroutine Used:**
None
- 6. On Return:**
DF is not changed by PUSHAC.
- 7. Length:**

CDP1801 version	CDP1802 version	bytes
21	19	

- 1. Subroutine:** POPAC
POP
- 2. Function:**
POPAC pops the top two bytes off the stack and stores them in R(AC).1 and R(AC).0. On return, R(MA) is left pointing to the byte below the top of the stack, i.e., $R(MA) = R(SP)+1$.
POP is another entry of the subroutine which pops two bytes off the stack without storing them in R(AC).
- 3. Calling Procedure:**
SEP CALL
A(POPAC) ... or POP
- 4. Registers Used:**
R(MA), R(AC) for POPAC only, R(SP).
- 5. Other Subroutine Used:**
None
- 6. On Return:**
DF = 0 means the top two bytes were not stored in R(AC).
DF = 1 means R(AC) contains the top two bytes of stack.
- 7. Length:**

	CDP1801 version	CDP1802 version	bytes
POPAC	29	26	
POP	25	23	bytes

- 1. Subroutine:** PUSHCQ
- 2. Function:**
PUSHCQ pushes the contents of R(CR), R(MA), R(MQ) onto stack. R(MQ).1, R(MQ).0, R(MA).1, R(MA).0, R(CR).1, R(CR).0 are stored from top to bottom accordingly. R(MA) is changed by PUSHCQ.
- 3. Calling Procedure:**
SEP CALL
A(PUSHCQ)
- 4. Registers Used:**
R(CR), R(MA), R(MQ), R(SP), R(MA) is changed by PUSHCQ.
- 5. Other Subroutine Used:**
None
- 6. On Return:**
DF is not changed by PUSHCQ.
- 7. Length:**

CDP1801 version	CDP1802 version	bytes
33	28	

- 1. Subroutine:** POPCQ
- 2. Function:**
POPCQ pops each of the top six bytes of the stack into R(MQ).1, R(MQ).0, R(MA).1, R(MA).0, R(CR).1, R(CR).0 in sequence.
- 3. Calling Procedure:**
SEP CALL
A(POPCQ)
- 4. Registers Used:**
R(MQ), R(MA), R(CR), R(SP).
- 5. Other Subroutine Used:**
None
- 6. On Return:**
DF is not changed by POPCQ.
- 7. Length:**

CDP1801 version	CDP1802 version	bytes
32	30	

1. Subroutine: **LOADOP**
LOAD
2. Function:
LOADOP and LOAD are entries to a subroutine that load the contents of memory (two bytes), whose high order byte is pointed to by R(MA), into R(AC). R(MA) is incremented twice by LOAD. LOADOP fetches the address of the operand and stores it into R(MA) and then does a LOAD. LOAD assumes R(MA) is pointing to operand.
3. Calling Procedure:

SEP CALL
A(LOADOP)

A(OPR) .. Load M(OPR) and M(OPR+1)
into R(AC)

SEP CALL
A(LOAD)

A(LOAD) .. Load M[R(MA)] and M[R(MA+1)]
into R(AC)

4. Registers Used:
R(LINK) for LOADOP only, R(MA), R(AC).
5. Other Subroutines Used:
LOADOP: should be followed by LOAD.
LOAD: none.
6. On Return:
DF is not changed by LOADOP or LOAD.
7. Length:

	CDP1801 version	CDP1802 version	
LOADOP:	9	9	bytes
LOAD:	5	5	bytes

8. Example:

.. assume M(OPR) contains X'AB'
.. assume M(OPR+1) contains X'CD'

SEP CALL
A(LOADOP)
A(OPR) .. address of ABCD
.. R(AC) now contains ABCD
.. R(MA) contains A(OPR+2)

1. Subroutine: **LODCON**
2. Function:
LODCON loads the two-byte constant following the subroutine call into register AC.
3. Calling Procedure:
SEP CALL
A(LODCON)
X'nnnn' .. the constant
[code]
4. Registers Used:
R(LINK), R(AC).
5. Other Subroutine Used:
None
6. On Return:
DF is not changed by LODCON.
7. Length:

CDP1801 version	CDP1802 version	
5	5	bytes

1. Subroutine:

STOROP
STORE

2. Function:

STOROP and STORE are entries of a subroutine which stores the contents of R(AC) into the memory locations pointed to by R(MA). On return, R(MA) is pointing to the next 16-bit word, i.e., R(MA) is incremented twice.

STOROP fetches the address of the operand and stores it into R(MA). It then does a STORE.

STORE assumes R(MA) contains the address of the operand.

3. Calling Procedure:

.. to enter from STOROP

SEP CALL

.A(STOROP)

.A(OPR) .. address of operand

[code] .. R(MA) is now A(OPR)+#02

.. to enter from STORE

SEP CALL

.A(STORE)

[code] .. R(MA) is now A(OPR)+#02

4. Registers Used:

R(MA), R(AC), R(LINK) for STOROP only.

5. Other Subroutines Used:

STOROP: should be followed in memory by STORE.

STORE: none.

6. On Return:

DF is not changed by STORE or STOROP.

7. Length:

	CDP1801 version	CDP1802 version	bytes
STOROP:	11	11	bytes
STORE:	7	7	bytes

1. Subroutine:

COMPOP
COMP

2. Function:

COMPOP and COMP are entries to a subroutine that compares R(AC) and the two-byte contents of memory pointed to by R(MA). DF is set to 1 if R(AC) is greater than or equal to the contents of memory.

COMPOP fetches the operand address and stores it into R(MA). It then does a COMP.

COMP assumes R(MA) is pointing to the operand.

3. Calling Procedure:

.. enter from COMPOP

SEP CALL

.A(COMPOP)

.A(OPR) .. address of operand

.. enter from COMP

SEP CALL

.A(COMP) .. do comparison

4. Registers Used:

R(AC), R(MA), R(LINK) for COMPOP only.

5. Other Subroutines Used:

COMPOP: should be followed in memory by COMP.

COMP: none.

6. On Return:

DF = 0 means $R(AC) < M(MA)$.

DF = 1 means $R(AC) \geq M(MA)$.

7. Number:

Representation: Signed 2's complement

Width: 16 bits

Range: $8000 \leq \text{number}_{16} \leq 7FFF$
 $-32768 < \text{number}_{10} \leq 32767$

8. Length:

	CDP1801 version	CDP1802 version	bytes
COMPOP:	22	22	bytes
COMP:	18	18	bytes

9. Example:

.. assume R(AC) = 0001;

.. M(OPR) contains X'FFFF'

SEP CALL

.A(COMPOP)

.A(OPR) .. address for comparison

.. DF is set to 1 since

.. 0001 > FFFF(= -1)

1. Subroutine: TEST
2. Function: TEST tests if R(AC) is equal to zero. If so, TEST returns to the 3rd byte immediately following the SEP call of TEST in the calling program. If R(AC) is not zero, TEST returns to the 5th byte following the SEP call of TEST in the calling program.

DF is set to 1 if AC is negative.

3. Calling Procedure:

SEP CALL

,A(TEST)

BR ZEROL

.. TEST returns here
if AC = 0.

[non-zero exit] .. TEST returns here
if AC ≠ 0.

4. Registers Used:

R(LINK), R(AC).

5. Other Subroutine Used:

None

6. On Return:

DF = 0 means number in R(AC) is positive.

DF = 1 means number in R(AC) is negative.

7. Length:

CDP1801

version

11

CDP1802

version

10

bytes

1. Subroutine: SWAPAQ

2. Function:

This subroutine exchanges the contents of R(AC) and R(MQ). R(CR).0 is used as a scratch register.

3. Calling Procedure:

SEP CALL

,A(SWAPAQ)

4. Registers Used:

R(CR), R(MA), R(AC).

5. Other Subroutine Used:

None

6. On Return:

DF is not changed by SWAPAQ.

7. Length:

CDP1801

version

13

CDP1802

version

13

bytes

Appendix A - Standard Subroutine Call and Return Technique

I. CDP1801 Version:

A. Subroutine CALL:

CALLR: SEP R3 .. to subroutine ...
 CALLS: GHI R6 .. save R6 on stack.
 STR R2
 DEC R2
 GLO R6
 STR R2
 DEC R2
 GHI R3 .. save R3 in R6.
 PHI R6
 GLO R3
 PLO R6
 LDA R6 .. set subroutine address
 PHI R3 .. in R3.
 LDA R6
 PLO R3
 BR CALLR .. jump to subroutine.

B. Subroutine RETURN:

RETR: SEP R3 .. return to main
 RET: GHI R6 .. restore return address
 PHI R3 .. into R3.
 GLO R6
 PLO R3
 INC R2 .. restore old R6 saved on stack
 LDA R2 .. into R6
 PLO R6
 LDA R2
 PHI R6
 DEC R2
 BR RETR .. return.

II. CDP1802 Version:

A. Subroutine CALL:

CALLR: SEP R3 .. to subroutine ...
 CALLS: SEX R2 .. point to stack
 GHI R6 .. save R6 on stack.
 STXD
 GLO R6
 STXD
 GHI R3 .. save R3 in R6.
 PHI R6
 GLO R3
 PLO R6
 LDA R6 .. set subroutine address
 PHI R3 .. in R3.
 LDA R6
 PLO R3
 BR CALLR .. jump to subroutine.

B. Subroutine RETURN:

RETR: SEP R3 .. return to main ...
 RET: GHI R6 .. restore return address
 PHI R3 .. into R3.
 GLO R6
 PLO R3
 SEX R2 .. restore old R6 saved
 INC R2 .. on stack into R6.
 LDXA
 PLO R6
 LDX
 PHI R6
 BR RETR .. return.

Appendix B(I) -

Arithmetic Subroutine

Listing for CDP1801

```

!M
0000 ; 0001      .... COPYRIGHT 1975 RCA CORPORATION ....
0000 ; 0002
0000 ; 0003      ORG #0400
0400 ; 0004      .. COSMAC ARITHMETIC SUBROUTINE PACKAGE
0400 ; 0005      ..
0400 ; 0006      .. EXTRACTED FOR BINARY ARITHMETIC
0400 ; 0007      .. AND CONVERSIONS FOR DECIMAL
0400 ; 0008      .. OCCUPIES 1K BYTES
0400 ; 0009      ..
0400 ; 0010      ..
0400 ; 0011      .. DESIGNED FOR STANDARD CSDP SUBROUTINE CALL&RETURN
0400 ; 0012      ..
0400 ; 0013      .. FOR STANDARD LINKAGE,
0400 ; 0014      SP= #02 ...IT SHOULD BE THE STACK POINTER.
0400 ; 0015      PC= #03 ...IT IS THE PROGRAM COUNTER
0400 ; 0016      .. USED BY THESE SUBROUTINES.
0400 ; 0017      CALL= #04 ...IT SHOULD POINT TO THE ROUTINE
0400 ; 0018      .. WHICH EFFECTS SUBROUTINE CALLS.
0400 ; 0019      RETN= #05 ...IT SHOULD POINT TO THE ROUTINE
0400 ; 0020      .. WHICH EFFECTS SUBROUTINE RETURN.
0400 ; 0021      LINK= #06 ...IT SHOULD POINT TO CALL PARAMETER
0400 ; 0022      .. .. (USUALLY OPERAND ADDRESSES AND/OR CONSTANT)
0400 ; 0023      ..
0400 ; 0024      ..
0400 ; 0025      .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0400 ; 0026      AR= #0A ..(USED FOR RESULT ADDRESS)
0400 ; 0027
0400 ; 0028      NR= #0B ..(USED FOR RESULT DIGIT COUNT)
0400 ; 0029      .. 16-BIT BINARY ARITHMETIC ROUTINES.
0400 ; 0030      .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0400 ; 0031      AC= #0F .. 16-BIT ACCUMULATOR=RF.
0400 ; 0032      MQ= #0E .. 16-BIT ACCUMULATOR=RE EXTENSION.
0400 ; 0033      MA= #0D .. (TEMPORARY) OPERAND MEMORY ADDRESS.
0400 ; 0034      CR= #0C .. (TEMPORARY) SCRATCHPAD AND COUNTER.
0400 ; 0035      ..
0400 ; 0036      ORG *+#08 ..FOR PAGE BOUNDARY
0400 ; 0037      ..
0400 ; 0038      .. SWAP AC WITH MQ REGISTERS
0400 ; 0039      SWAPQ: GHI MQ .. SAVE MQ.1
0400 ; 0040      PLO CR .. IN CR.0 (COULD HAVE PUSHED ON STAC
0400 ; 0041      GHI AC .. NOW AC.1 TO MQ.1
0400 ; 0042      PHI MQ
0400 ; 0043      GLO MQ .. SAVE MQ.0
0400 ; 0044      PHI AC .. IN AC.1
0400 ; 0045      GLO AC .. THEN AC.0 TO MQ.0
0400 ; 0046      PLO MQ
0400 ; 0047      GHI AC .. NOW SAVED MQ.0 TO AC.0
0400 ; 0048      PLO AC
0400 ; 0049      GLO CR .. FINALLY SAVED MQ.1
0400 ; 0050      PHI AC .. TO AC.1
0400 ; 0051      SEP RETN
0400 ; 0052      .. 16-BIT SUBTRACT AC FROM CONSTANT
0400 ; 0053      .. ****AC= CONSTANT-AC
0400 ; 0054      .. ***** (TO CALL, WRITE) *****
0400 ; 0055      .. ****CALL SDCON ; ,CONSTANT
0400 ; 0056      SDCON: GLO LINK .. (ESSENTIALLY SAME AS ADCON)
0400 ; 0057      PLO MA

```



```

0417 96:      0058      GHI LINK
0418 8D:      0059      PHI MA
0419 16:      0060      INC LINK
041A 16:      0061      INC LINK
041B 3021:    0062      BR SD
041D :        0063      ..
041D :        0064      .. 16-BIT SUBTRACT AC FROM OPERAND
041D :        0065      .. ****AC=OPRN-AC
041D :        0066      .. ***** (TO CALL, WRITE) *****
041D :        0067      .. ****CALL SDOP ; ,A(OPRN)
041D :        0068      ..
041D 46:      0069      SDOP: LDA LINK .. FETCH OPERAND ADDRESS
041F 8D:      0070      PHI MA .. TO MA REGISTER
041F 46:      0071      LDA LINK
0420 8D:      0072      PLO MA .. FALL INTO SD
0421 :        0073      .. 16-BIT SUBTRACT AC FROM OPERAND
0421 :        0074      .. CALL HERE IF OPERAND
0421 :        0075      .. ADDRESS IN REGISTER MA
0421 :        0076      .. ****AC=M(R(MA))-AC
0421 :        0077      .. ***** (TO CALL, WRITE) *****
0421 :        0078      .. ****CALL SD
0421 :        0079      ..
0421 8D:      0080      SD:  SEX MA .. SET X PTR TO MA
0422 9F:      0081      GHI AC ..CHECK SIGN BIT OF AC
0423 F3:      0082      XOR  ..AND OPERAND MA
0424 FA80:    0083      ANI #80 ..RESULT A 1 IF DIFF
0426 52:      0084      STR SP  ..AND STORE ON STACK
0427 1D:      0085      INC MA  ..POINT TO LOW 8
0428 8F:      0086      GLO AC  .. FETCH AC LOW 8
0429 F5:      0087      SD  .. SUBTRACT FROM LOW 8 IN MEMORY
042A AF:      0088      PLO AC  .. PUT IT BACK
042B 2D:      0089      DEC MA  .. NOW HIGH 8
042C 9F:      0090      GHI AC
042D 3335:    0091      RDE SDNB .. (NO BORROW)
042F FC01:    0092      ADI #01 .. PROPAGATE LOW 8 BORROW
0431 3A35:    0093      RNF SDNB
0433 F4:      0094      ADD  .. SECOND BORROW; FORCE BORROW OUT
0434 3A:      0095      ,#3A .. AND SKIP OVER REGULAR SUBTRACT
0435 F5:      0096      SDNB: SD .. SUBTRACT HIGH 8
0436 8F:      0097      PHI AC  .. PUT IT BACK
0437 42:      0098      LDA SP  ..LOAD STORED COMPARING BIT OF OPERANDS
0438 22:      0099      RFC SP  ..RESET STACK POINTER
0439 323D:    0100      RZ SDFF ..IF OPERAND'S SIGNS ARE SAME
043A 9F:      0101      GHI AC  ..NO OVERFLOW POSSIBLE
043C F3:      0102      XOR  ..OTHER WISE CHECK RESULT
043D FC80:    0103      SDFF:  ADI #80 ..SET DF=0 IF OK
043F 05:      0104      SEP RETN ..RETURN
0440 :        0105      ..
0440 :        0106      .. 16-BIT SUBTRACT FROM AC (ADDRESS IN CALL)
0440 :        0107      .. ****AC=AC-OPRN
0440 :        0108      .. ***** (TO CALL, WRITE) *****
0440 :        0109      .. ****CALL SMOP ; ,A(OPRN)
0440 :        0110      ..
0440 46:      0111      SMOP: LDA LINK
0441 8D:      0112      PHI MA
0442 46:      0113      LDA LINK
0443 AD:      0114      PLO MA
0444 :        0115      .. 16-BIT SUBTRACT FROM AC (ADDRESS IN MA)

```

```

0444 ; 0116 .. CALL HERE IF OPERAND
0444 ; 0117 .. ADDRESS IN REGISTER MA
0444 ; 0118 .. ****AC=AC-M(R(MA))
0444 ; 0119 .. ***** (TO CALL, WRITE) *****
0444 ; 0120 .. ****CALL SM
0444 ; 0121 ..
0444 FD; 0122 SM: SEX MA ..SET X PTR
0445 9F; 0123 GHI AC ..GET SIGN OF AC AND
0446 F6; 0124 SHR ..AND STORE IN 7TH BIT OF CR
0447 52; 0125 STR SP ..BIT PUT IN (SP) FIRST
0448 9F; 0126 GHI AC ..AND SEE IF OPERANDS SIGNS ARE THE SAME
0449 F3; 0127 XOR
044A F80; 0128 ANI #80 ..TAKE OUT COMPARING SIGN BIT
044C F2; 0129 SEX SP ..NOW STORE THAT BIT IN 8TH OF CR
044D F4; 0130 ADD ..BY ADDING TO IT
044F 52; 0131 STR SP ..AND STORE THESE TWO BITS ON STACK
044F 1D; 0132 INC MA ..POINT TO LOW 8
0450 FD; 0133 SEX MA ..REMEMBER TO SET X TO OPERANDS
0451 AF; 0134 GLO AC .. FETCH AC LOW 8
0452 F7; 0135 SM .. SUBTRACT MEMORY FROM IT
0453 AF; 0136 PLO AC .. PUT IT BACK.
0454 2D; 0137 DEC MA .. NOW HIGH 8
0455 9F; 0138 GHI AC
0456 335F; 0139 RDF SMNB
0458 FF0; 0140 SMI #01 .. PROPAGATE BORROW OF LOW 8
045A 335F; 0141 RDF SMNB
045C F3; 0142 XOR .. SECOND BORROW; FORCE BORROW OUT,
045D 38; 0143 ,#3A .. WHILE SUBTRACTING HIGH 8.
045F F7; 0144 SMNB: SM .. HIGH 8 SUBTRACT, NO BORROW ACROSS.
045F RF; 0145 PHI AC .. PUT HIGH 8 BACK
0460 F2; 0146 SEX SP ..NOW CHECK IF UNDERFLOWED
0461 F0; 0147 LDH ..LOAD THE STORED TWO BITS
0462 F80; 0148 ANI #80 ..AND TAKE OUT THE COMPARING SIGN BIT
0464 326B; 0149 RZ SM1 ..THE SAME, UNDERFLOW NOT POSSIBLE
0466 9F; 0150 GHI AC ..OTHERWISE HAVE TO COMPARE SIGN OF RE
.SULT
0467 F6; 0151 SHR ..WITH SIGN OF AC
0468 F3; 0152 XOR ..SIGN BIT OF AC WAS STORED ON STACK
0469 F80; 0153 ANI #40 ..TAKE OUT THAT 7TH BIT
046B F6; 0154 SM1: SHR ..SET DF=0 IF OK
046C 3270; 0155 RZ SMPT ..THE SAME, IT'S OK
046E FF0; 0156 SMI #00 ..OTHERWISE SET DF=1
0470 05; 0157 SMPT: SEP RETN ..RETURN WITH DF=NO BORROW
0471 ; 0158 ..
0471 ; 0159 ..
0471 ; 0160 ..16X16 BIT SIGNED MULTIPLY(2'S COMPLEMENT)
0471 ; 0161 ..****AC=AC*OPRN
0471 ; 0162 ..***** (TO CALL, WRITE) *****
0471 ; 0163 ..****CALL MPYOP ; ,A(OPRN)
0471 ; 0164 ..
0471 46; 0165 MPYOP: LDA LINK ..FETCH MULTIPLICAND ADDR
0472 BD; 0166 PHI MA ..INTO REGISTER A
0473 46; 0167 LDA LINK
0474 AD; 0168 PLO MA ..FALL INTO MPY
0475 ; 0169 .. 16X16 BIT SIGNED MULTIPLY (2'S COMPLEMENT
0475 ; 0170 .. CALL HERE IF OPERAND ADDRESS
0475 ; 0171 .. IN REGISTER MA

```

0475 :	0172	.. ****AC=AC*M(R(MA))
0475 :	0173	.. ***** (TO CALL, WRITE) *****
0475 :	0174	.. ****CALL MPY
0475 :	0175	..
0475 F01	0176	MPY: SEX MA .. SET X NOW
0476 9F1	0177	GHI AC ..CHECK IS THE SIGN OF MULTICAND
0477 F31	0178	XOR ..THE SAME AS THE SIGN OF MULTIPR
0478 FA801	0179	ANI #80 ..AND STORE THAT BIT
047A BC1	0180	PHI CP ..INTO CR.1
047B FA101	0181	LDI #10 .. SET COUNTER TO 16
047D AC1	0182	PLO CR
047F FA001	0183	LDI #00 .. INITIALIZE MQ TO 0
0480 BF1	0184	PHI MQ ... TO HOLD PRODUCT.
0481 AF1	0185	PLO MQ
0482 2C1	0186	MOL: DEC CR ..IF NOT, DECREMENT IT.
0483 9F1	0187	GHI AC .. SHIFT AC (=MULTIPLIER) RIGHT
0484 F61	0188	SHR
0485 BF1	0189	PHI AC
0486 BF1	0190	GLO AC
0487 3B801	0191	RNF *+#06
0489 F61	0192	SHR .. SHIFT 1 ACROSS BYTES ..
048A F9801	0193	ORI #80
048C 381	0194	,#38
048D F61	0195	SHR .. SHIFT 0 ACROSS BYTES ..
048F AF1	0196	PLO AC
048F 9F1	0197	GHI MQ
0490 3B981	0198	RNF MPB ..IF NO BIT OUT, DON'T ADD.
0492 1D1	0199	INC MA .. POINT TO LOW 8 OF MULTIPLICAND
0493 AC1	0200	GLO CR .. IF NOT LAST ITERATION,
0494 3A0F1	0201	RNZ MPA .. GO ADD.
0496 D41	0202	SEP CALL ..CALL SUBTRACT MQS(MA) ROUTINE
0497 04081	0203	,A(DSM)
0499 F01	0204	SFX MA ..FIX X PTR
049A 9C1	0205	GHI CR ..NOW ARE THE SIGNS OF OPERANDS THE SAME
049B FC801	0206	MPB: ADI #80 ..TEST FOR SIGN BIT
049D 30A81	0207	RR MPS+#05 ..IF NEGATIVE, SIGN EXTEND
049F D41	0208	MPA: SEP CALL ..CALL ADD MQ & (MA) SUM
04A0 04F81	0209	,A(DVA)
04A2 F01	0210	SFX MA
04A3 33A81	0211	MPS: RDF *+#05
04A5 F01	0212	LDX
04A6 FC801	0213	ADI #80..IF OPERAND IS NEGATIVE,THEN
04A8 9F1	0214	GHI MQ ..PUT MQ.1 INTO D
04A9 3BAF1	0215	RNF *+#06 ..EXTEND A 1 FOR -SIGN BIT
04AB F61	0216	SHR .. SHIFT IN 1
04AC F9801	0217	ORI #80
04AF 381	0218	,#38
04AF F61	0219	SHR .. SHIFT IN 0
04B0 BF1	0220	PHI MQ
04B1 BF1	0221	GLO MQ .. CONTINUE TO LOW 8 OF MQ
04B2 3B881	0222	RNF *+#06
04B4 F61	0223	SHR
04B5 F9801	0224	ORI #80
04B7 381	0225	,#38
04B8 F61	0226	SHR
04B9 AF1	0227	PLO MQ
04BA 3BC01	0228	RNF MT .. IF NO CARRY OUT, ITERATE.
04BC 9F1	0229	GHI AC .. ADD CARRY OUT INTO AC MSR.
04BD F9801	0230	ORI #80


```

04BF BF1
04C0 BC1
04C1 3A21
04C3 9F1
04C4 FCA01
04C6 BF1
04C7 3ACB1
04C9 FBFF1
04CB 3AD21
04CD 9F1
04CE 3BD21
04D0 FBFF1
04D2 FCFF1
04D4 D51
04D5 1
04D5 1D1
04D6 3FE81
04D8 BF1
04D9 F71
04DA AF1
04DB 2D1
04DC 9F1
04DD 33F51
04DE FF011
04F1 33F51
04E3 F31
04E4 381
04E5 F71
04E6 30F61
04F8 BF1
04F9 F41
04FA AF1
04FB 2D1
04FC 9F1
04FD 3AF51
04FE FC011
04F1 3AF51
04F3 F51
04F4 381
04F5 F41
04F6 BF1
04F7 D51
04FA 1
04FB F21
04FC BF1
04FA 521
04FB F41
04FC AF1
04FD 9F1
04FE 521
04FF 3B031
0501 FC011
0503 F41
0504 BF1
0505 D51
0506 1
0506 1
0506 1

0231 PHI AC
0232 MT: GLO CR ..CHECK COUNTER
0233 BNZ MPL ..IF COUNTER IS NOT 0,GO BACK FOR MORE
0234 MPX: GHI AC .. FINISHED;
0235 ADI #80 .. CHECK FOR PRODUCT > 15 BITS.
0236 GLO MQ
0237 BNF *+04 ..THAT'S HIGH 17 BITS
0238 XPI #FF .. ALL 00 OR FF.
0239 RNZ *+07 .. NAW.
0240 GHI MQ
0241 BNF *+04
0242 XRI #FF
0243 ADI #FF .. SET DF IF PRODUCT > 15 BITS
0244 SEP RETN .. RETURN.
0245 ...SUBROUTINE TO BE INCLUDED IN DIV,DIVQ,MPY ....
0246 DARS: INC MA ..MQ.0-(MA)
0247 BNF DVA ..IF NO FLAG, MQ.+(MA)
0248 DSM: GLO MQ ..LOAD MQ.0 TO D
0249 SM ..MQ.0-(MA.0)
0250 PLO MQ ..AND STORE BACK INTO MQ.0
0251 DEC MA ..DO THE SAME FOR MQ.1
0252 GHI MQ ..LOAD MQ.1 INTO D
0253 RDF *+08 ..IF NO BORROW ,SKIP
0254 SMI #01 ..MINUS 1 FOR BORROW OUT
0255 RDF *+04 ..IF MQ.1 NO BORROW OUT, SKIP
0256 XOR ..OTHERWISE ONLY DO XOR
0257 ,#38 ..AND SKIP OVER NORMAL SUBTRACTION
0258 SM ..MQ.1-(MA.1)
0259 RR DVB ..SKIP OVER ADD STEPS
0260 DVA: GLO MQ ..MQ+(MA)
0261 ADD ..MQ.0+(MA.0)
0262 PLO MQ ..AND PUT BACK INTO MQ.0
0263 DEC MA ..DO THE SAME FOR MQ.1 &MA.1
0264 GHI MQ ..LOAD MQ.1 INTO D
0265 BNF *+08 ..IF NO CARRY OUT,SKIP CARRY OVER
0266 ADI #01 ..CARRY OVER STEP
0267 BNF *+04 ..SKIP CARRY IF NO CARRY RESULT
0268 SD ..OTHERWISE IT'S 0 HERE,PUT IT BACK
0269 ,#38 ..AND SKIP NORMAL ADDITION
0270 ADD ..ADD MQ.1 AND (MA.1)
0271 DVB: PHI MQ ..AND STORE INTO MQ.1
0272 SEP RETN ..
0273 ... SUBROUTINE TO BE INCLUDED IN DIV,DIVQ ....
0274 DSHL: SFX SP ..SET X TO STACK
0275 GLO MQ ..ADD MQ.0 TO ITSELF.
0276 STR SP ..CAUSE MQ.0 SHIFT TO LEFT
0277 ADD ..
0278 PLO MQ ..AND STORE BACK
0279 GHI MQ ..DO THE SAME FOR MQ.1
0280 STR SP
0281 BNF *+04 ..IF NO CARRY OUT,SKIP CARRY OVER
0282 ADI #01 ..ADD 1 FOR CARRY OVER
0283 ADD ..ADD MQ.1 TO ITSELF
0284 PHI MQ ..STORE BACK
0285 SEP RETN ..
0286 .. 32/16 BIT SIGNED DIVIDE (2'S COMPLMENT)
0287 .. AC=MQ,AC/OPRN
0288 .. QUOTIENT IN AC , REMAINDER IN MQ

```

```

0506 ; 0289 .. ***** (TO CALL, WRITE) *****
0506 ; 0290 .. CALL DIVOP ; ,A(OPRN)
0506 ; 0291 ..
0506 46; 0292 DIVOP: LDA LINK .. FETCH OPERAND ADDRESS
0507 RD; 0293 PHI MA .. TO REGISTER MA
0508 46; 0294 LDA LINK
0509 AD; 0295 PLO MA .. FALL INTO DIVO/DIV/DIVQ.
050A ; 0296 .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
050A ; 0297 ... OPTION #1: CLEAR MQ AND CHECK FOR ZERO D
050A ; 0298 .. ****AC=MQ,AC/M(R(MA))
050A ; 0299 .. ***** (TO CALL, WRITE) *****
050A ; 0300 .. ****CALL DIVO
050A FD; 0301 DIVO: SEX MA .. SET X TO POINT TO DIVISOR (n)
050B 9F; 0302 GHI AC .. LOOK AT AC SIGN (0)
050C FC80; 0303 ADI #80 .. COPY IT TO DF (0)
050F FA00; 0304 LDI #00 .. EXTEND #00 IF POSITIVE, (0)
0510 3B14; 0305 RNF *#04 .. (0)
0512 FAF; 0306 LDI #FF .. #FF IF NEGATIVE. (0)
0514 BF; 0307 PHI MQ .. GIVING +n OR -n IN MQ (n)
0515 AF; 0308 PLO MQ .. (n)
0516 4D; 0309 LDA MA .. CHECK FOR ZERO DIVISOR (0)
0517 F1; 0310 OR .. (0)
0518 2D; 0311 DEC MA .. (DON'T FORGET TO FIX POINTER) (n)
0519 FD00; 0312 SDI #00 .. IF ZERO, CALL IT DIVIDE CHECK(0)
051B 3B1F; 0313 RNF DIV ..GO ON IF NO DIVIDE CHECK ERR
051D 05; 0314 SFP RETN ..AND RETURN WITH DF=1 (0)
051F ; 0315 ... OPTION #2: PERMIT 32-BIT DIVIDEND; (/)
051F ; 0316 ... MAKE SURE QUOTIENT DOES NOT EXCEED 16 BITS (/)
051F ; 0317 .. ***** (TO CALL, WRITE) *****
051E ; 0318 ..***** CALL DIV
051F 9F; 0319 DIV: GHI MQ ..SAVE PARTIAL DIVIDEND(/)
051F FC; 0320 PHI CR ..INTO CR.1(/)
0520 BF; 0321 GLO MQ ..(/)
0521 AC; 0322 PLO CR ..AND CR.n(/)
0522 D4; 0323 SEP CALL ..CALL SHIFT LEFT ROUTINE(/)
0523 04FB; 0324 ,A(DSHL) ..(/)
0525 9F; 0325 GHI AC ..ALSO SHIFT HIGH OF AC(/)
0526 FAF; 0326 ANI #80 ..(WHICH IS THE 16TH BIT)(/)
0528 322C; 0327 RZ D7 ..BUT IF 0,SKIP SHIFT(/)
052A 1F; 0328 INC MQ ..IF 1, SHIFT INTO MQ(/)
052B 3B; 0329 ,#38 ...(/)
052C FC; 0330 D7: GLO CR ..SEE IF MQ.0 =0(/)
052D 3A34; 0331 BNZ D2 ..IF NOT GO THROUGH CHECKING STEPS(/)
052F 9C; 0332 GHI CR ..SEE IF MQ.1 IS #40(/)
0530 FB40; 0333 XRI #40 ..WHICH SHOULD RESULT 0 IF TRUE(/)
0532 323D; 0334 RZ D4 ..IF TRUE,SKIP NORMAL CHECKING(/)
0534 9C; 0335 D2: GHI CR ..CHECK IF HIGH 2 BITS OF MQ ARE THE SAME(/)
0535 FAF; 0336 ANI #C0 ..TAKE OUT 2 HIGH BITS(/)
0537 FD00; 0337 SDI #00 ..SEE IF THEY ARE THE SAME(/)
0539 FC80; 0338 ADI #80 ..IF NOT,HIGH BIT IS 1(/)
053B 33C0; 0339 RNF DVXX ..SET DF AND RETURN(/)
053D 9C; 0340 D4: GHI CR ..LOOK AT THE SIGNS OF DIVND AND DIVSR(/)
053E FD; 0341 SEX MA ..POINT X TO DIVSR(/)
053F F3; 0342 XOR ..(/)
0540 FAF; 0343 ANI #80 ..IF DIFFERENT, RESULT 80(/)
0542 FD00; 0344 SDI #00 ..SET DF IF SAME(/)
0544 D4; 0345 SEP CALL ..CALL ABS(MQ-(MA)) SUB(/)

```

0545 0405:	0346	.A(DARS) ..(//)
0547 FD:	0347	SEX MA ..SET X TO DIVSOR(//)
0548 9C:	0348	GHI CR ..LOOK AT THE SIGN OF DIVND(//)
0549 FCR0:	0349	ADI #80 ..DF=1 IF DIVND IS NEG(//)
054B 3R5C:	0350	RNF DV2 ..IF NOT, DON'T COMPLMENT DIFFERENCE(//)
054D 9F:	0351	GHI MQ ..COMPLEMENT MQ.1(//)
054F FD00:	0352	SDI #00 ..BY SUBTRACTING FROM 0(//)
0550 RF:	0353	PHI MQ ..STORE BACK INTO MQ.1(//)
0551 AF:	0354	GLO MQ ..DO THE SAME FOR MQ.0(//)
0552 325C:	0355	RZ DV2 ..BUT IF 0, NO NEED TO(//)
0554 9F:	0356	GHI MQ ..(//)
0555 FF01:	0357	SMI #01 ..OTHERWISE DO BORROW FROM MQ.1(//)
0557 RF:	0358	PHI MQ ..(//)
0558 AF:	0359	GLO MQ ..DO COMPLMENTATION FOR MQ.0(//)
0559 FD00:	0360	SDI #00 ..BY SUBTRACTING FROM 0(//)
055B AF:	0361	PLO MQ ..AND PUT BACK INTO MQ.0(//)
055C 9F:	0362 DV2:	GHI MQ ..NOW LOOK AT THE DIFF OF MQ&(MA)(//)
055D 3265:	0363	RZ D10 ..IF 0, CHECK MQ=0 SPECIAL CASES(//)
055F FCR0:	0364	ADI #80 ..CHECK IF MQ IS NEGATIVE(//)
0561 33C5:	0365	RDF DDQ ..IF YES, NO PROBLEM(//)
0563 30RF:	0366	RR D9 ..IF NOT 0 NOR NEG, DIVND IS TOO LARGE(//)
0565 RF:	0367 D10:	GLO MQ ..HERE WE CHECK DIFF=0 CASES(//)
0566 FCFE:	0368	ADI #FE ..IF MQ.0 IS NOT EITHER 0 OR 1(//)
0568 33C0:	0369	RDF DVXX ..THEN DIVND IS STILL TOO LARGE(//)
056A FFEF:	0370	XRI #FF ..RESULT 0 IF MQ.0 WAS 1(//)
056C 3AA1:	0371	RNZ DVH ..IF NOT, MUST BE 0, GO TO DVH(//)
056F 9C:	0372	GHI CR ..SEE IF DIVND IS NEGATIVE(//)
056F FA80:	0373	ANI #80 ..RESULT 80 IF YES(//)
0571 32FE:	0374	RZ D9 ..IF DIVND POSITIVE, IT CANNOT DIVIDE(//)
0573 F4:	0375	ADD ..SEE IF DIVISOR IS POSITIVE(//)
0574 33C0:	0376	RDF DVXX ..IF NEGATIVE, DIVND CANNOT DIVIDE(//)
0576 9F:	0377	GHI AC ..NOW CHECK IF AC+(MA)>0 IF LOW AC IS 0(//)
0577 F4:	0378	ADD ..AC.1+(MA.1)(//)
0578 RF:	0379	PHI MQ ..AND STORE SUM IN MQ.1(//)
0579 1D:	0380	INC MA ..DO THE SAME FOR LOW AC & (MA.0)(//)
057A RF:	0381	GLO AC ..LOAD AC.0 INTO 0(//)
057B F4:	0382	ADD ..AC.0 TO (MA.0)(//)
057C AF:	0383	PLO MQ ..(//)
057D 3A83:	0384	RNF *+06 ..SKIP CARRY OVER IF NO CARRY OUT(//)
057F 9F:	0385	GHI MQ ..DO CARRY OVER(//)
0580 FCR0:	0386	ADI #01 ..BY ADD 1 TO MQ.1(//)
0582 RF:	0387	PHI MQ ..(//)
0583 F0:	0388	LDX ..NOW CHECK LOW BIT (MA) IS 0 OR 1(//)
0584 F6:	0389	SHR ..SHIFT THAT LOW BIT OUT(//)
0585 2D:	0390	DEC MA ..REMEMBER TO RESET MA(//)
0586 9F:	0391	GHI MQ ..READY TO ADD #80(//)
0587 3RAC:	0392	RNF *+05 ..IF LOW BIT OF AC IS 1(//)
0589 FCR0:	0393	ADI #80 ..TO MQ.1(//)
058B RF:	0394	PHI MQ ..SAVE STATUS(//)
058C FCR0:	0395	ADI #80 ..SEE IF MQ > 0(//)
058F 9C:	0396	GHI CR ..SEE IF DIVND IS POSITIVE(//)
058F FA80:	0397	ANI #80 ..BY TAKING OUT THE SIGN BIT(//)
0591 3A97:	0398	RNZ *+06 ..IF NOT, GO THROUGH DIFF CHECK(//)


```

0593 38BE:      0399      BNF D9  ..IF AC+(MA) >=0, BAD(//)
0595 30C5:      0400      BR DDQ  ..OTHERWISE IS OK(//)
0597 9F:        0401      GHI MQ  ..MQ=0?(//)
0598 33C0:      0402      BDF DVXX ..IF NEGATIVE, THEN OUT(//)
059A 3AC5:      0403      BNZ DDQ  ..IF NOT, THEN NO PROBLEM(//)
059C 8F:        0404      GLO MQ  ..MAKE SURE MQ.0 IS 0 TOO(//)
059D 32BE:      0405      RZ D9  ..IF YES, THEN RETURN WITH DF=1(//)
059F 30C5:      0406      BR DDQ  ..OR ELSE GO TO DIVIDE(//)
05A1 9C:        0407 DVH:   GHI CR  ..SEE IF THE OPERANDS SIGNS DIFF(//)
05A2 F3:        0408      XOR  ..BY COMPARING SIGN BITS(//)
05A3 F8A0:      0409      ANI #80  ..AND TAKE OUT THAT BIT(//)
05A5 3A76:      0410      BNZ D10+#11 ..IF SIGNS DIFF, IT'S OK(//)
05A7 9C:        0411      GHI CR  ..OTHERWISE TEST SIGN OF DIVND(//)
05A8 FC80:      0412      ADI #80  ..IF POSITIVE, RETURN WITH DF=1(//)
05AA 38BF:      0413      BNF D9  ..RETURN WITH DF=1(//)
05AC 1D:        0414      INC MA  ..SEE IF LOW BIT OF AC IS 0 OR 1(//)
05AD F0:        0415      LDX  ..LOAD THAT IN D(//)
05AF 2D:        0416      DEC MA  ..REMEMBER TO RESET MA(//)
05AF F6:        0417      SHR  ..SHIFT THAT BIT OUT(//)
05B0 8F:        0418      GLO AC  ..IF AC IS NOT 0, NOT PROBLEM(//)
05B1 3AC5:      0419      BNZ DDQ  ..GO TO DIVIDE(//)
05B3 9F:        0420      GHI AC  ..READY TO CHECK #80 IF LOW BIT AC IS 1(//)
05B4 33BA:      0421      RDF *+#06  ..(//)
05B6 32BF:      0422      RZ D9  ..NO GOOD, RETURN WITH DF=1(//)
05B8 30C5:      0423      BR DDQ  ..ANY THING ELSE IS OK(//)
05BA F8A0:      0424      XRI #80  ..IF AC.1 IS'NT #80(//)
05BC 3AC5:      0425      BNZ DDQ  ..IT'S OK(//)
05BE FF00:      0426 D9:   SMI #00  ..DF IS SET TO 1(//)
05C0 9C:        0427 DVXX:  GHI CR  ..PUT ORGINAL DIVND(//)
05C1 BF:        0428      PHI MQ  ..INTO MQ(//)
05C2 8C:        0429      GLO CR  ..(//)
05C3 AF:        0430      PLO MQ  ..(//)
05C4 D5:        0431      SEP RETN ..AND RETURN WITH DF=1(//)
05C5 9C:        0432 DDQ:   GHI CR  ..PUT ORGINAL DIVND(//)
05C6 BF:        0433      PHI MQ  ..BACK INTO MQ(//)
05C7 8C:        0434      GLO CR  ..(//)
05C8 AF:        0435      PLO MQ  ..(//)
05C9 :          0436      ... OPTION #3: ASSUME BENIGN PROGRAM; NOCHECK
05C9 :          0437      .. ***** (TO CALL, WRITE) *****
05C9 :          0438      .. ****CALL DIVQ
05C9 9F:        0439 DIVQ:  GHI MQ  ..LOOK AT DIVIDEND SIGN
05CA FC80:      0440      ADI #80  .. IF POSITIVE,
05CC F890:      0441      LDI #90  .. PLAN TO BEGIN WITH SUBTRACT,
05CE 38D2:      0442      BNF *+#04 .. (ALSO SAVE SIGN OF DIVIDEND)
05D0 F850:      0443      LDI #50  .. OTHER WISE BEGIN WITH ADD.
05D2 AC:        0444      PLO CR  .. SET ITERATION COUNT IN CR.0
05D3 D4:        0445 DVL:   SEP CALL ..CALL LEFT SHIFT SUB
05D4 04F8:      0446      ,A(DSHL)
05D6 8F:        0447      GLO AC  .. NOW DO AC
05D7 52:        0448      STR SP  ..
05D8 F4:        0449      ADD
05D9 AF:        0450      PLO AC
05DA 9F:        0451      GHI AC
05DB 52:        0452      STR SP  ..
05DC 38F4:      0453      BNF *+#08
05DE FC01:      0454      ADI #01
05E0 38E4:      0455      BNF *+#04
05E2 F5:        0456      SD  .. DON'T LOSE CARRY OUT
05E3 38:        0457      ,#38

```

05E4 F4:	0458	ADD
05E5 RF:	0459	PHI AC
05F6 3RF9:	0460	BNF *+H03 .. BIT SHIFTED OUT OF AC.1;
05F8 1F:	0461	INC MQ .. GOES INTO MQ.0
05F9 FD:	0462	SEX MA ..
05FA AC:	0463	GLO CR .. NOW, WAS THAT ADD, OR SUBTRACT?
05EB F3:	0464	XOR .. IT DEPENDS ON SAVED FLAG,
05FC FCR0:	0465	ADI #80 .. AND SIGN OF DIVISOR.
05EE D4:	0466	SEP CALL ..CALL ABS(MQ-(MA)) SUB
05EF 04D5:	0467	,A(DABS)
05F1 FD:	0468	SEX MA
05F2 2C:	0469	DEC CR .. COUNT DOWN ITERATION COUNTER
05F3 AC:	0470	GLO CR
05F4 FA7F:	0471	ANI #7F
05F6 3RFB:	0472	BNF *+H05 .. TEST CARRY OUT OF ADD/SUBTRACT
05F8 1F:	0473	INC AC .. IF 1, SHIFT INTO QUOTIENT,
05F9 F980:	0474	ORI #80 .. AND FLAG NEXT OP AS SUBTRACT.
05FA AC:	0475	PLO CR .. OTHERWISE IT'S ADD.
05FC FA3F:	0476	ANI #3F .. LOOK AT COUNTER:
05FE 3AD3:	0477	RNZ DVL .. IF NOT 0, LOOP BACK;
0600 3320:	0478	RDF DVR .. AT END, CHECK REMAINDER ADJUST.
0602 3A:	0479	,#38
0603 1F:	0480	DVC: INC AC .. (FINAL DIVIDE STEP)
0604 F3:	0481	XOR .. BE SURE TO GET POLARITY
0605 FCR0:	0482	ADI #80 .. OF ADJUSTMENT RIGHT...
0607 1D:	0483	INC MA .. YES, ADD DIVISOR BACK ON,
0608 RF:	0484	GLO MQ .. TO CORRECT FOR FINAL SUBTRACT,
0609 3316:	0485	RDF DVM .. (ADDING NEGATIVE IS SUBT.)
060B F4:	0486	ADD .. WHICH SHOULDN'T HAVE.
060C AF:	0487	PLO MQ
060D 2D:	0488	DEC MA
060F 9F:	0489	GHI MQ
060F 3R13:	0490	BNF *+H04
0611 FC01:	0491	ADI #01 .. NO NEED TO SAVE CARRY OUT
0613 F4:	0492	ADD
0614 301F:	0493	RR DVR-#01
0616 F7:	0494	DVM: SM .. SAME THING,
0617 AF:	0495	PLO MQ .. EXCEPT, FOR NEGATIVE DIVISOR.
0618 2D:	0496	DEC MA
0619 9F:	0497	GHI MQ
061A 331F:	0498	RDF *+H04
061C FF01:	0499	SMI #01
061E F7:	0500	SM
061F RF:	0501	PHI MQ
0620 9E:	0502	DVR: GHI MQ .. IF REMAINDER IS NOT ZERO,
0621 3A26:	0503	RNZ *+H05
0623 RF:	0504	GLO MQ
0624 3234:	0505	RZ DVM .. BUT IT IS; NO PROBLEM.
0626 AC:	0506	GLO CR .. IF NOT ZERO,
0627 FA40:	0507	ANI #40 .. IT SHOULD BE SAME SIGN
0629 FCC0:	0508	ADI #C0 .. AS ORIGINAL DIVIDEND.
062B 9F:	0509	GHI MQ
062C 3330:	0510	RDF *+H04
062F FB80:	0511	XRI #80
0630 FCR0:	0512	ADI #80 .. IF NOT, WE NEED
0632 3B03:	0513	BNF DVC .. ONE MORE DIVIDE ITERATION.
0634 F0:	0514	DVN: LDX .. FINALLY, IF DIVISOR NEGATIVE,
0635 FCR0:	0515	ADI #80
0637 3B44:	0516	BNF DVX .. (IT'S NOT; WE ARE DONE)
0639 RF:	0517	GLO AC .. COMPLEMENT QUOTIENT,

063A FBFF;	0518	XRI #FF .. BY INVERTING IT
063C AF;	0519	PLO AC
063D 9F;	0520	GHI AC
063E FBFF;	0521	XRI #FF
0640 BF;	0522	PHI AC
0641 1F;	0523	INC AC .. THEN INCREMENTING
0642 FC00;	0524	ADI #00 .. ALSO CLEAR DF.
0644 D5;	0525	DVX: SEP RETN .. DF=0 IF DIVIDE SUCCESSFUL.
0645 ;	0526	..
0645 ;	0527	.. BINARY TO DECIMAL CONVERSION
0645 ;	0528	.. ****DECIMAL NUMBER = AC
0645 ;	0529	.. DECIMAL NUMBER = SIGN,N0,N1,N2,N3,..
0645 ;	0530	.. SIGN=#0B +
0645 ;	0531	.. SIGN=#0D -
0645 ;	0532	.. N0=10**0 DIGIT
0645 ;	0533	.. N1=10**1 DIGIT, ETC
0645 ;	0534	.. ***** (TO CALL, WRITE) *****
0645 ;	0535	.. ****CALL CBD ; ,A(NUMBER) ; LENGTH
0645 46;	0536	CRD: LDA LINK .. GET DECIMAL ADDRESS
0646 BA;	0537	PHI AR
0647 46;	0538	LDA LINK
0648 AA;	0539	PLO AR
0649 46;	0540	LDA LINK .. GET SIZE (#)
064A AB;	0541	PLO NR
064B BB;	0542	PHI NR
064C FB00;	0543	LDI #00 .. STORE MINUS (+)
064F 5A;	0544	STR AR
064F 0F;	0545	GHI AC .. SET DF IF POSITIVE
0650 FD7F;	0546	SDI #7F
0652 FA;	0547	SEX AR
0653 D4;	0548	SEP CALL .. NOW CHANGE SIGN TO MATCH
0654 06A0;	0549	,A(DZS)
0656 3350;	0550	RDF *+ #07
0658 D4;	0551	SEP CALL .. SET AC POSITIVE
0659 0415;	0552	,A(SDCON)
065B 00;	0553	,#00
065C 00;	0554	,#00
065D 04;	0555	SEP CALL .. PUSH AC INTO STACK,
065E 06F9;	0556	,A(PUSHAC)
0660 FB00;	0557	LDI #00 .. ONLY TO REPLACE IT
0662 5D;	0558	STR MA .. BY A DIVISOR OF 10
0663 1D;	0559	INC MA
0664 FB0A;	0560	LDI #0A
0666 5D;	0561	STR MA
0667 2D;	0562	DFC MA
0668 2B;	0563	CBL: DEC NP .. DECREMENT DIGIT COUNT
0669 AB;	0564	GLO NR .. CHECK END
066A 327F;	0565	RZ CRX .. ZERO IS OVERFLOW.
066C FB00;	0566	LDI #00
066F AF;	0567	PLO MQ
066F BF;	0568	PHI MQ
0670 D4;	0569	SEP CALL .. DIVIDE BY 10
0671 05C9;	0570	,A(DIVQ)
0673 1A;	0571	INC AR .. (<)
0674 BE;	0572	GLO MQ .. REMAINDER IS THIS DIGIT
0675 5A;	0573	STR AR
0676 BF;	0574	GLO AC .. CHECK FOR NON-ZERO
0677 3A68;	0575	RNZ CBL
0679 9F;	0576	GHI AC
067A 3A68;		

067A 3A68:	0577	BNZ CBL .. CONTINUE IF MORE DIGITS
067C F6:	0578	SHR .. CLFAR DF FOR SUCCESS
067D 38:	0579	,#38
067E F5:	0580	CRX: SD .. SET DF FOR FAILURE
067F 12:	0581	INC SP .. POP STACK (THAT TEN)
0680 12:	0582	INC SP
0681 D5:	0583	SEP RETN .. DONE.
0682 :	0584	..
0682 :	0585	..
0682 :	0586	..
0682 :	0587	.. RESTORE RESULT POINTER
0682 3B85:	0588	DRP: RNF *+#03 .. SAVE FLAG IN NR.0
0684 1R:	0589	INC NR .. (ASSUME WAS 0)
0685 F2:	0590	SEX SP .. PUSH SAVED DIGIT COUNT
0686 9R:	0591	GHI NR .. INTO STACK
0687 52:	0592	STR SP
0688 8A:	0593	GLO AR .. SUBTRACT FROM RESULT ADDRESS
0689 F7:	0594	SM .. (<)
068A AA:	0595	PLO AR
068B 3391:	0596	BDF *+#06 .. (=)(<)
068D 9A:	0597	GHI AR .. BORROW FROM HIGH BYTE (=)
068F FF01:	0598	SMI #01 .. (=)(<)
0690 RA:	0599	PHI AR .. (=)
0691 8R:	0600	GLO NR .. RECOVER DF
0692 F6:	0601	SHR
0693 F0:	0602	LDX .. POP SAVED COUNT,
0694 AR:	0603	PLO NR .. INTO NR.0
0695 D5:	0604	SEP RETN .. RETURN
0696 :	0605	..
0696 5A:	0606	DZR: STR AR .. STORE A BYTE
0697 1A:	0607	INC AR .. ADVANCE TO NEXT (<)
0698 2R:	0608	DEC NR .. CHECK COUNT
0699 8R:	0609	GLO NR
069A 3282:	0610	RZ DRP .. AT 0, GO RESTORE POINTERS
069C F800:	0611	LDI #00 .. CONTINUE STORING 0
069F 3096:	0612	RR DZR
06A0 :	0613	.. COMPARE SIGNS, AND CLEAR RESULT (ADDRESS
06A0 F80R:	0614	DZS: LDI #0B .. IF DF=0, TRY "+" (+)
06A2 3BA6:	0615	RNF *+#04
06A4 F800:	0616	LDI #0D .. OTHERWISE "-" (+)
06A6 F3:	0617	XOR .. COMPARE TO PRESENT M(R(X))
06A7 32AR:	0618	RZ *+#04
06A9 F806:	0619	LDI #06 .. NOT EQUAL, SO CONDITION MINUS(+)
06AB F80R:	0620	XRI #0B .. IF EQUAL SET PLUS (+)
06AD 3096:	0621	RR DZR .. GO STORE, WITH ZERO IN RESULT.
06AF :	0622	..
06AF :	0623	.. 16-BIT ADD TO AC, OPERAND ADDRESS IN CALL
06AF :	0624	.. ****AC=AC+OPRN .. OPRN=2 BYTE OPERAND
06AF :	0625	.. ***** (TO CALL, WRITE) *****
06AF :	0626	.. ****CALL ADDOP , A(OPRN)
06AF :	0627	..
06AF :	0628	..
06AF 46:	0629	ADDOP: LDA LINK .. FETCH OPERAND ADDRESS
06B0 RD:	0630	PHI MA .. TO REGISTER MA
06B1 46:	0631	LDA LINK
06B2 AD:	0632	PLO MA .. FALL INTO ADD
06B3 :	0633	.. 16-BIT ADD TO AC, OPERAND ADDRESS IN REGI
06B3 :	0634	.. CALL HERE IF OPRN ADDRESS
06B3 :	0635	.. IS IN REGISTER MA
06B3 :	0636	.. ****AC=AC+M(R(MA))

```

06R3 :      0637      .. ***** (TO CALL, WRITE) *****
06R3 :      0638      .. *****CALL ADD
06R3 :      0639      ..
06R3 FD:      0640      ADD: SEX MA ..CHECK SIGN BIT OF AC
06R4 9F:      0641      GHI AC ..GET THE OPERAND
06R5 F3:      0642      XOR ..AND OPERAND QMA
06R6 FAR0:    0643      ANI #80 ..RESULT A 1 IF DIFF
06R8 52:      0644      STR SP ..STORE ON STACK
06R9 1D:      0645      INC MA ..POINT TO LOW 8 BITS
06RA 8F:      0646      GLO AC .. FETCH AC LOW 8
06RB F4:      0647      ADD .. ADD LOW 8 FROM MEMORY
06RC AF:      0648      PLO AC .. PUT IT BACK
06RD 2D:      0649      DEC MA .. POINT TO HIGH 8 MEMORY LOCATION
06RE 9F:      0650      GHI AC
06RF 3RC7:    0651      BNF ADDNC
06C1 FC01:    0652      ADI #01 .. ADD IN CARRY OUT OF LOW 8
06C3 3RC7:    0653      RNF ADDNC .. (OMIT 3 LINES IF CARRY OUT NOT
06C5 F5:      0654      SD .. OOPS, THAT CARRIED OUT, LEAVING 0:
06C6 38:      0655      ,#38 .. FINISH ADD, FORCING CARRY TO 1.
06C7 F4:      0656      ADDNC: AND .. ADD HIGH 8
06C8 8F:      0657      PHI AC .. PUT IN AC.
06C9 42:      0658      LDA SP ..LOAD THE STORED COMPARING SIGN BIT
06CA 22:      0659      DEC SP ..RESET STACK PTR
06CB 3AD6:    0660      BNZ ADDRT ..NOT POSSIBLE
06CD 9F:      0661      GHI AC ..OTHERWISE SEE IF SUM IS RIGHT
06CE F3:      0662      XOR ..BY COMPARING SIGN BITS
06CF FAR0:    0663      ANI #80 ..TAKE OUT THE COMPARING BIT
06D1 32D6:    0664      BZ ADDRT ..IF SAME, THEN NOT OVERFLOWED
06D3 FCFF:    0665      ADI #FF ..OTHERWISE, SET DF=1
06D5 38:      0666      ,#38
06D6 F6:      0667      ADDRT: SHR ..SHIFT OUT 0 INTO DF
06D7 D5:      0668      SEP RETN ..RETURN TO MAIN
06D8 :      0669      ..
06D8 :      0670      .. 16-BIT CONSTANT ADD TO AC.
06D8 :      0671      .. CALL HERE FOR ADD CONSTANT TO AC
06D8 :      0672      .. ****AC=AC+CONSTANT
06D8 :      0673      .. ***** (TO CALL, WRITE) *****
06D8 :      0674      .. *****CALL ADDCON ; ,CONSTANT
06D8 :      0675      ..
06D8 86:      0676      ADDCON: GLO LINK .. COPY LINK TO MA
06D9 AD:      0677      PLO MA
06DA 96:      0678      GHI LINK
06DB BD:      0679      PHI MA
06DC 16:      0680      INC LINK .. INCREMENT PAST DATUM
06DD 16:      0681      INC LINK
06DE 30B3:    0682      BR ADD .. GO ADD.
06E0 :      0683      ..
06E0 :      0684      .. 16-BIT ADD FROM TOP OF STACK
06E0 92:      0685      ADDST: GHI SP .. COPY STACK POINTER
06E1 BD:      0686      PHI MA .. TO MA REGISTER
06E2 82:      0687      GLO SP
06E3 AD:      0688      PLO MA
06E4 1D:      0689      INC MA .. ADVANCE TO SUB-TOP (D)
06E5 1D:      0690      INC MA .. (D)
06E6 1D:      0691      INC MA .. (D)
06E7 30B3:    0692      BR ADD .. GO DO IT
06E9 :      0693      ..
06E9 :      0694      ..
06E9 :      0695      ..
06E9 :      0696      .. PUSH AC INTO STACK

```

06E9 :	0697	.. STACK POINTER = SP
06E9 :	0698	.. ***** (TO CALL, WRITE) *****
06F9 :	0699	.. *****CALL PUSHAC
06F9 :	0700	.. PUSH AC (UNDER TOP OF STACK)
06F9 92:	0701	PUSHAC: GHI SP .. COPY STACK POINTER TO MA
06FA RD:	0702	PHI MA
06FA R2:	0703	GLO SP
06FC AD:	0704	PLO MA
06FD 1D:	0705	INC MA .. NOW SLICE OFF TOP 2 BYTES,
06FE 1D:	0706	INC MA
06FF ED:	0707	SEX MA
06F0 F0:	0708	LDX .. TO MAKE A 2-BYTE HOLE.
06F1 52:	0709	STR SP
06F2 2D:	0710	DEC MA
06F3 22:	0711	DEC SP
06F4 F0:	0712	LDX
06F5 52:	0713	STR SP
06F6 9F:	0714	GHI AC .. NOW STUFF AC INTO THE HOLE.
06F7 5D:	0715	STR MA
06F8 RF:	0716	GLO AC
06F9 1D:	0717	INC MA
06FA 5D:	0718	STR MA
06FA 2D:	0719	DEC MA .. LEAVE MA POINTING TO IT.
06FC 22:	0720	DEC SP
06FD D5:	0721	SEP RETN .. (AC UNCHANGED)
06FE :	0722	.. PUSH CR, MA, MQ (UNDER TOP OF STACK)
06FE RD:	0723	PUSHCQ: GLO MA .. FIRST PUSH MA ONTO TOP
06FF 52:	0724	STR SP
0700 22:	0725	DEC SP
0701 9D:	0726	GHI MA
0702 52:	0727	STR SP
0703 92:	0728	GHI SP .. NOW COPY SP TO MA
0704 RD:	0729	PHI MA
0705 R2:	0730	GLO SP
0706 AD:	0731	PLO MA
0707 22:	0732	DEC SP
0708 1D:	0733	INC MA .. THEN ADJUST IT
0709 1D:	0734	INC MA
070A 1D:	0735	INC MA .. TO POINT INTO OLD TOP
070B RF:	0736	GLO MQ .. CONTINUE PUSHING, MQ
070C 52:	0737	STR SP
070D 22:	0738	DEC SP
070E 9F:	0739	GHI MQ
070F 52:	0740	STR SP
0710 22:	0741	DEC SP
0711 FD:	0742	SEX MA .. NOW COPY OLD TOP TO NEW TOP
0712 F0:	0743	LDX
0713 52:	0744	STR SP
0714 22:	0745	DEC SP
0715 2D:	0746	DEC MA
0716 F0:	0747	LDX
0717 52:	0748	STR SP
0718 22:	0749	DEC SP
0719 9C:	0750	GHI CR .. FINALLY INSERT CR IN HOLE
071A 5D:	0751	STR MA
071B 1D:	0752	INC MA
071C 8C:	0753	GLO CR
071D 5D:	0754	STR MA
071E D5:	0755	SEP RETN .. (MA IS GARBAGE OUT)
071F :	0756	..

071F :	0757	.. POP STACK INTO MQ, MA, CR (UNDER TOP OF S
071F 12:	0758	POPCQ: INC SP
0720 92:	0759	GHI SP .. COPY STACK POINTER TO MA
0721 RD:	0760	PHI MA
0722 82:	0761	GLO SP
0723 AD:	0762	PLO MA
0724 1D:	0763	INC MA .. ADJUST POINTER TO CR DATUM
0725 1D:	0764	INC MA
0726 1D:	0765	INC MA
0727 1D:	0766	INC MA
0728 1D:	0767	INC MA
0729 1D:	0768	INC MA
072A 4D:	0769	LDA MA .. FETCH IT
072B BC:	0770	PHI CR
072C 4D:	0771	LDA MA
072D AC:	0772	PLO CR
072F 2D:	0773	DEC MA .. COPY TOP OF STACK INTO GAP
072F 2D:	0774	DEC MA
0730 42:	0775	LDA SP
0731 5D:	0776	STR MA
0732 1D:	0777	INC MA
0733 42:	0778	LDA SP
0734 5D:	0779	STR MA
0735 42:	0780	LDA SP .. THEN POP MQ
0736 RF:	0781	PHI MQ
0737 42:	0782	LDA SP
0738 AF:	0783	PLO MQ
0739 42:	0784	LDA SP .. FINALLY POP MA
073A RD:	0785	PHI MA
073B 42:	0786	LDA SP
073C AD:	0787	PLO MA
073D 22:	0788	DEC SP
073F D5:	0789	SEP RETN
073F :	0790	.. 16-BIT ACCUMULATOR LOAD (ADDRESS IN CALL)
073F :	0791	.. *****AC=OPRN
073F :	0792	.. ***** (TO CALL, WRITE) *****
073F :	0793	.. *****CALL LOADOP ; ,A(OPRN)
073F :	0794	..
073F 46:	0795	LOADOP: LDA LINK .. FETCH ADDRESS
0740 RD:	0796	PHI MA .. TO MA REGISTER
0741 46:	0797	LDA LINK
0742 AD:	0798	PLO MA .. FALL INTO LOAD
0743 :	0799	.. 16-BIT ACCUMULATOR LOAD (ADDRESS IN MA)
0743 :	0800	.. CALL HERE IF OPERAND
0743 :	0801	.. ADDRESS IN REGISTER MA
0743 :	0802	.. *****AC=M(R(MA))
0743 :	0803	.. ***** (TO CALL, WRITE) *****
0743 :	0804	.. *****CALL LOAD
0743 :	0805	..
0743 4D:	0806	LOAD: LDA MA .. FETCH HIGH 8
0744 RF:	0807	PHI AC
0745 4D:	0808	LDA MA .. NOW LOW 8
0746 AF:	0809	PLO AC .. LEAVE MA AT NEXT DOUBLE-BYTE
0747 D5:	0810	SFP RETN .. GEE, THAT WAS QUICK.
0748 :	0811	..
0748 :	0812	.. 16-BIT ACCUMULATOR LOAD FROM CONSTANT IN
0748 :	0813	.. *****AC=CONSTANT
0748 :	0814	.. ***** (TO CALL, WRITE) *****
0748 :	0815	.. *****CALL LODCON ; ,CONSTANT
0748 :	0816	..

```

0748 46: 0817 LODCON: LDA LINK .. FETCH HIGH 8
0749 RF: 0818 PHI AC
074A 46: 0819 LDA LINK .. THEN LOW 8
074B AF: 0820 PLO AC
074C D5: 0821 SEP RETN
074D : 0822 ..
074D : 0823 .. 16-BIT ACCUMULATOR STORE (ADDRESS IN CALL
074D : 0824 .. ****OPRN=AC
074D : 0825 .. ***** (TO CALL, WRITE) *****
074D : 0826 .. ****CALL STOROP : ,A(OPRN)
074D : 0827 ..
074D 46: 0828 STOROP: LDA LINK .. FETCH ADDRESS INTO MA
074E RD: 0829 PHI MA
074F 46: 0830 LDA LINK
0750 AD: 0831 PLO MA .. THEN FALL INTO STORE
0751 : 0832 .. 16-BIT ACCUMULATOR STORE (ADDRESS IN MA)
0751 : 0833 .. ****M(R(MA))=AC
0751 : 0834 .. ***** (TO CALL, WRITE) *****
0751 : 0835 .. ****CALL STORE
0751 : 0836 ..
0751 9F: 0837 STORE: GHI AC .. FIRST HIGH 8
0752 5D: 0838 STR MA
0753 1D: 0839 INC MA .. INCREMENT MA, SINCE STR DOESN'T
0754 AF: 0840 GLO AC .. NOW LOW 8
0755 5D: 0841 STR MA
0756 1D: 0842 INC MA .. LEAVE MA POINTING TO NEXT WORD
0757 D5: 0843 SEP RETN .. QUIT
0758 : 0844 ..
0758 : 0845 ..
0758 : 0846 .. 16-BIT COMPARE, OPERAND ADDRESS IN CALL
0758 : 0847 .. ****AC-OPRN (DF SET IF 0 OR +)
0758 : 0848 .. ***** (TO CALL, WRITE) *****
0758 : 0849 .. ****CALL COMOP : ,A(OPRN)
0758 : 0850 ..
0758 46: 0851 COMPOP: LDA LINK .. FETCH ADDRESS
0759 RD: 0852 PHI MA .. TO MA REGISTER
075A 46: 0853 LDA LINK
075B AD: 0854 PLO MA
075C : 0855 .. 16-BIT COMPARE, OPERAND ADDRESS IN REGIST
075C : 0856 .. CALL HERE IF OPERAND
075C : 0857 .. ADDRESS IN REGISTER MA
075C : 0858 .. ****AC-M(R(MA)) (DF SET IF 0 OR +)
075C : 0859 .. ***** (TO CALL, WRITE) *****
075C : 0860 .. ****CALL COMP
075C : 0861 ..
075C FD: 0862 COMP: SEX MA .. COMPARE HIGH 8 FRST
075D 9F: 0863 GHI AC ..CHECK IF SIGN OF OPERANDS ARE SAME
075F F3: 0864 XOR ..BY LOOKING AT THE HIGHEST SIGN BIT
075F FA8D: 0865 ANI #80 ..RESULT A '1' IF NEGATIVE
0761 3A6C: 0866 BNZ CNE ..IF NOT, THEN GO TO CNE
0763 9F: 0867 GHI AC
0764 F7: 0868 SM
0765 3A6D: 0869 BNZ CMPX .. NOT EQUAL QUIT
0767 1D: 0870 INC MA .. TRY LOW 8
0768 AF: 0871 GLO AC
0769 F7: 0872 SM
076A 2D: 0873 DEC MA .. LEAVE MA POINTING TO IT
076B 3A: 0874 ,#38
076C F4: 0875 CNE: ADD ..SEE IF OPERAND IS NEGATIVE
076D D5: 0876 CMP

```

0760 D5:	0876	CMPX: SEP RETN ..DF=1 IF AC>=(MA)
076E :	0877	..
076E :	0878	..
076F :	0879	.. DECIMAL TO BINARY CONVERSION
076E :	0880	.. ***AC=DECIMAL NUMBER OF N BYTES
076E :	0881	.. DECIMAL NUMBER = SIGN,N0,N1,N2,N3 ...
076E :	0882	.. SIGN = #0B +
076E :	0883	.. SIGN = #0D -
076F :	0884	.. N0=10**0 DIGIT
076F :	0885	.. N1=10**1 DIGIT, ETC
076E :	0886	.. ***** (TO CALL, WRITE) *****
076E :	0887	.. ****CALL CDB ; ,A(NUMBER) ; LENGTH
076F :	0888	..
076F 46:	0889	CDB: LDA LINK .. GET DECIMAL ADDRESS
076F BA:	0890	PHI AR
0770 46:	0891	LDA LINK
0771 AA:	0892	PLO AR
0772 46:	0893	LDA LINK .. AND SIZE (#)
0773 AB:	0894	PLO NR
0774 2B:	0895	DEC NR ..MINUS ONE FOR SIGN
0775 FA:	0896	SEX AR .. LOOK AT SIGN (!)
0776 4A:	0897	LDA AR .. (<)
0777 F80D:	0898	XRI #0D .. IS IT MINUS? (+)
0779 327D:	0899	CDS: RZ *+#04 .. #00 IF YES,
077B F8FF:	0900	LDI #FF .. #FF IF NO:
077D BB:	0901	PHI NR .. REMEMBER THAT
077F F800:	0902	LDI #00 .. INITIALIZE AC TO 0
0780 AF:	0903	PLO AC
0781 BF:	0904	PHI AC
0782 8B:	0905	GLO NR .. GO TO OTHER END
0783 52:	0906	STR SP .. BY ADDING SIZE
0784 E2:	0907	SEX SP
0785 AA:	0908	GLO AR .. TO ADDRESS
0786 F4:	0909	ADD .. (<)
0787 AA:	0910	PLO AR
0788 3BFF:	0911	RNF CDL .. (=)(<)
078A 9A:	0912	GHI AR .. PROPAGATE CARRY (=)
078B FC01:	0913	ADI #01 .. ACROSS BYTES (=)(<)
078D BA:	0914	PHI AR .. (=)
078F 8B:	0915	CDL: GLO NR .. ANY MORE DIGITS
078F 32C1:	0916	RZ CDX .. NO
0791 2B:	0917	DEC NR .. YES.
0792 D4:	0918	SEP CALL .. MULTIPLY AC * 10
0793 06F9:	0919	,A(PUSHAC)
0795 D4:	0920	SEP CALL .. FROM STACK
0796 074B:	0921	,A(LODCON)
0798 00:	0922	,#00
0799 0A:	0923	,#0A
079A D4:	0924	SEP CALL
079B 0475:	0925	,A(MPY)
079D 3BA2:	0926	RNF *+#05 ..IF NOT OVERFLOWED, SKIP EXIT
079F 12:	0927	INC SP ..RESET STACK
07A0 12:	0928	INC SP ..
07A1 D5:	0929	SEP RETN ..OVERFLOW EXIT
07A2 D4:	0930	SEP CALL .. SAVE PRODUCT IN STACK
07A3 0751:	0931	,A(STORE)
07A5 2A:	0932	DEC AR .. (<)
07A6 4A:	0933	LDA AR .. NOW GET DIGIT
07A7 2A:	0934	DEC AR
07A8 FA0F:	0935	ANI #0F .. REMOVE OVERBITS

07AA AF1	0936	PLO AC
07AB FA001	0937	LDI #00
07AD RF1	0938	PHI AC
07AF 041	0939	SEP CALL .. ADD TO PREVIOUS NUMBER
07AF 06F01	0940	,A(ADDST)
07R1 121	0941	CDO: INC SP
07R2 121	0942	INC SP
07B3 3BFF1	0943	RNF CDL .. ADVANCE TO NEXT DIGIT
07R5 RR1	0944	GLO NR ..NO MORE DIGIT?
07B6 3AC01	0945	RNZ CDRT ..OVERFLOWED
07R8 9R1	0946	GHI NR ..POSITIVE NUMBER?
07B9 3AC01	0947	RNZ CDRT ..OVERFLOWED
07RR 2F1	0948	DEC AC ..IF AC IS #8000 IS OK
07RC 9F1	0949	GHI AC ..TEST THAT BIT
07RD FC801	0950	ADI #80 ..
07RF 1F1	0951	INC AC ..REMEMBER TO RESET AC
07C0 D51	0952	CDRT: SEP RETN ..OVERFLOW RETURN
07C1 9R1	0953	CDX: GHI NR ..NO. LOOK AT SIGN
07C2 3AC91	0954	RNZ *+07 .. POSITIVE
07C4 041	0955	SEP CALL
07C5 04151	0956	,A(SDCON) .. NEGATIVE. SUBTRACT FROM 0
07C7 001	0957	,#00
07C8 001	0958	,#00
07C9 FC001	0959	ADI #00 .. CLEAR DF
07CB D51	0960	SEP RETN .. DF=0 IF OK.
07CC 1	0961	..
07CC 1	0962	..TEST 16-BIT ACCUMULATOR SIGN/ZERO
07CC 161	0963	INC LINK ..SKIP OVER NON ZERO RETURN
07CD 161	0964	INC LINK ..
07CF D51	0965	SEP RETN
07CF 9F1	0966	TEST: GHI AC .. FIRST LOOK AT SIGN
07D0 FC801	0967	ADI #80 ..SET DF IF MINUS
07D2 9F1	0968	GHI AC .. NOW CHECK FOR 0
07D3 3AC01	0969	RNZ TEST-#03 .. NO.
07D5 8F1	0970	GLO AC
07D6 32CF1	0971	RZ TEST-#01 .. GO TAKE ZERO EXIT.
07D8 30CC1	0972	BR TEST-#03 .. GO TAKE NON-ZERO EXIT.
07DA 1	0973	.. POP STACK (INTO AC) (UNDER TOP)
07DA 1	0974	.. POPS 2 BYTES FROM STACK INTO AC
07DA 1	0975	.. ***** (TO CALL, WRITE) *****
07DA 1	0976	.. *****CALL POPAC
07DA FF001	0977	POPAC: SMI #00 .. SET F TO REMEMBER THIS ENTRY
07DC 30FF1	0978	BR *+04
07DF FC001	0979	POP: ADI #00 .. CLEAR DF FOR THIS ENTRY
07E0 121	0980	INC SP
07F1 021	0981	GHI SP .. COPY SP TO MA
07E2 RD1	0982	PHI MA
07E3 821	0983	GLO SP
07E4 AD1	0984	PLO MA
07E5 1D1	0985	INC MA .. ADJUST TO SUB-TOP OF STACK
07F6 1D1	0986	INC MA
07F7 3BFF1	0987	RNF *+08
07F9 4D1	0988	LDA MA .. POPPING INTO AC. GET DATUM
07FA RF1	0989	PHI AC
07FB 4D1	0990	LDA MA
07FC AF1	0991	PLO AC
07FD 2D1	0992	DEC MA .. RESTORE POINTER
07EE 2D1	0993	DEC MA
07FF 421	0994	LDA SP .. NOW CLOSE UP THE GAP
07F0 5D1	0995	STR MA
07F1 1D1	0996	INC MA
07F2 421	0997	LDA SP
07F3 5D1	0998	STR MA
07F4 1D1	0999	INC MA
07F5 221	1000	DEC SP
07F6 D51	1001	SEP RETN .. MA POINTS TO NEW SUB-TOP
07F7 1	1002	END
0000		


```

0000 : 0001
0000 : 0002
0000 : 0003      ..COPYRIGHT 1976 RCA CORPORATION
0000 : 0004      .. COSMAC ARITHMETIC SUBROUTINE PACKAGE
0000 : 0005      ..
0000 : 0006      .. VERSION 1.1 FOR CDP1802
0000 : 0007      ..
0000 : 0008      .. EXTRACTED FOR BINARY ARITHMETIC
0000 : 0009      .. AND CONVERSIONS FOR DECIMAL
0000 : 0010      .. OCCUPIES 1K BYTES
0000 : 0011      ..
0000 : 0012      ..
0000 : 0013      .. DESIGNED FOR STANDARD COSP SUBROUTINE CALL & RETURN
0000 : 0014      ..
0000 : 0015      .. FOR STANDARD LINKAGE,
0000 : 0016      SP= #02 ...IT SHOULD BE THE STACK POINTER.
0000 : 0017      PC= #03 ...IT IS THE PROGRAM COUNTER
0000 : 0018      .. USED BY THESE SUBROUTINES.
0000 : 0019      CALL= #04 ...IT SHOULD POINT TO THE ROUTINE
0000 : 0020      .. WHICH EFFECTS SUBROUTINE CALLS.
0000 : 0021      RETN= #05 ...IT SHOULD POINT TO THE ROUTINE
0000 : 0022      .. WHICH EFFECTS SUBROUTINE RETURN.
0000 : 0023      LINK= #06 ...IT SHOULD POINT TO CALL PARAMETER
0000 : 0024      .. .. (USUALLY OPERAND ADDRESSES AND/OR CONSTANT)
0000 : 0025      ..
0000 : 0026      ..
0000 : 0027      .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0000 : 0028      AR= #0A ..(USED FOR RESULT ADDRESS)
0000 : 0029
0000 : 0030      NR= #0B ..(USED FOR RESULT DIGIT COUNT)
0000 : 0031      .. 16-BIT BINARY ARITHMETIC ROUTINES
0000 : 0032      .. THE FOLLOWING REGISTERS MUST BE ASSIGNED
0000 : 0033      AC= #0F .. 16-BIT ACCUMULATOR=RF.
0000 : 0034      MQ= #0F .. 16-BIT ACCUMULATOR=RE EXTENSION.
0000 : 0035      MA= #0D .. (TEMPORARY) OPERAND MEMORY ADDRESS.
0000 : 0036      CR= #0C .. (TEMPORARY) SCRATCHPAD AND COUNTER.
0000 : 0037      ..
0000 : 0038      ..
0000 : 0039      .. SWAP AC WITH MQ REGISTERS
0000 : 0040      SWAPAQ: GHI MQ .. SAVE MQ.1
0001 AC: 0041      PLO CR .. IN CR.0 (COULD HAVE PUSHED ON STAC
0002 QF: 0042      GHI AC .. NOW AC.1 TO MQ.1
0003 RF: 0043      PHI MQ
0004 RF: 0044      GLO MQ .. SAVE MQ.0
0005 RF: 0045      PHI AC .. IN AC.1
0006 RF: 0046      GLO AC .. THEN AC.0 TO MQ.0
0007 AF: 0047      PLO MQ
0008 QF: 0048      GHI AC .. NOW SAVED MQ.0 TO AC.0
0009 AF: 0049      PLO AC
000A RF: 0050      GLO CR .. FINALLY SAVED MQ.1
000B RF: 0051      PHI AC .. TO AC.1

```



```

000C D5:      0052      SEP RETN
000D :      0053      .. 16-BIT SUBTRACT AC FROM CONSTANT
000D :      0054      .. ****AC= CONSTANT-AC
000D :      0055      .. ***** (TO CALL, WRITE) *****
000D :      0056      .. ****CALL SDCON ; ,CONSTANT
000D 86:      0057      SDCON: GLO LINK .. (ESSENTIALLY SAME AS ANCON)
000F AD:      0058      PLO MA
000F 96:      0059      GHI LINK
0010 RD:      0060      PHI MA
0011 16:      0061      INC LINK
0012 16:      0062      INC LINK
0013 3019:    0063      RR SD
0015 :      0064      ..
0015 :      0065      .. 16-BIT SUBTRACT AC FROM OPERAND
0015 :      0066      .. ****AC=OPRN-AC
0015 :      0067      .. ***** (TO CALL, WRITE) *****
0015 :      0068      .. ****CALL SDOP ; ,A(OPRN)
0015 :      0069      ..
0015 46:      0070      SDOP: LDA LINK .. FETCH OPERAND ADDRESS
0016 RD:      0071      PHI MA .. TO MA REGISTER
0017 46:      0072      LDA LINK
0018 AD:      0073      PLO MA .. FALL INTO SD
0019 :      0074      .. 16-BIT SUBTRACT AC FROM OPERAND
0019 :      0075      .. CALL HERE IF OPERAND
0019 :      0076      .. ADDRESS IN REGISTER MA
0019 :      0077      .. ****AC=M(R(MA))-AC
0019 :      0078      .. ***** (TO CALL, WRITE) *****
0019 :      0079      .. ****CALL SD
0019 :      0080      ..
0019 FD:      0081      SD:  SFX MA .. SET X PTR TO MA
001A 9F:      0082      GHI AC ..CHECK SIGN BIT OF AC
001B F3:      0083      XOR  ..AND OPERAND 0MA
001C 52:      0084      STR SP  ..AND STORE ON STACK
001D 1D:      0085      INC MA  ..POINT TO LOW 8
001E AF:      0086      GLO AC  .. FETCH AC LOW 8
001F F5:      0087      SD  .. SUBTRACT FROM LOW 8 IN MEMORY
0020 AF:      0088      PLO AC  .. PUT IT BACK
0021 2D:      0089      DEC MA  .. NOW HIGH 8
0022 9F:      0090      GHI AC
0023 75:      0091      SDNR: SDR  .. SUBTRACT HIGH 8
0024 BF:      0092      PHI AC  .. PUT IT BACK
0025 02:      0093      LDN SP  ..LOAD STORED COMPARING BIT OF OPERANDS
0026 FF:      0094      SHL  ..CHECK STORED SIGN COMPARISON BIT
0027 3B2C:    0095      RNF SDFF ..IF OPERAND'S SIGNS ARE SAME
0029 9F:      0096      GHI AC  ..NO OVERFLOW POSSIBLE
002A F3:      0097      XOR  ..OTHER WISE CHECK RESULT
002B FF:      0098      SHL  ..SET DF=0 IF OK
002C D5:      0099      SDFF:  SEP RETN  ..RETURN
002D :      0100      ..
002D :      0101      .. 16-BIT SUBTRACT FROM AC (ADDRESS IN CALL)
002D :      0102      .. ****AC=AC-OPRN
002D :      0103      .. ***** (TO CALL, WRITE) *****
002D :      0104      .. ****CALL SMOP ; ,A(OPRN)
002D :      0105      ..
002D 46:      0106      SMOP: LDA LINK
002F BD:      0107      PHI MA
002F 46:      0108      LDA LINK
0030 AD:      0109      PLO MA
0031 :      0110      .. 16-BIT SUBTRACT FROM AC (ADDRESS IN MA)

```

0031 :	0111	.. CALL HERE IF OPERAND
0031 :	0112	.. ADDRESS IN REGISTER MA
0031 :	0113	.. ****AC=AC-M(R(MA))
0031 :	0114	.. ***** (TO CALL, WRITE) *****
0031 :	0115	.. ****CALL SM
0031 :	0116	..
0031 F01	0117 SM:	SFX MA ..SET X PTR
0032 9F1	0118	GHI AC ..GET SIGN OF AC AND
0033 F61	0119	SHR ..AND STORE IN 7TH BIT OF CR
0034 521	0120	STR SP ..PUT PUT IN (SP) FIRST
0035 9F1	0121	GHI AC ..AND SEE IF OPERANDS SIGNS ARE THE SAME
0036 F31	0122	XOR
0037 FA801	0123	ANI #80 ..TAKE OUT COMPARING SIGN BIT
0039 F21	0124	SFX SP ..NOW STORE THAT BIT IN 8TH OF CR
003A F41	0125	ADD ..BY ADDING TO IT
003B 521	0126	STR SP ..AND STORE THESE TWO BITS ON STACK
003C 1D1	0127	INC MA ..POINT TO LOW 8
003D F01	0128	SEX MA ..REMEMBER TO SET X TO OPERANDS
003E 8F1	0129	GLO AC .. FETCH AC LOW 8
003F F71	0130	SM .. SUBTRACT MEMORY FROM IT
0040 AF1	0131	PLO AC .. PUT IT BACK.
0041 2D1	0132	DEC MA .. NOW HIGH 8
0042 9F1	0133	GHI AC
0043 771	0134	SMNR: SMR .. HIGH 8 SUBTRACT, NO BORROW ACROSS.
0044 BF1	0135	PHI AC .. PUT HIGH 8 BACK
0045 F21	0136	SEX SP ..NOW CHECK IF UNDERFLOWED
0046 F01	0137	LNX ..LOAD THE STORED TWO BITS
0047 FF1	0138	SHL ..AND TAKE OUT THE COMPARING SIGN BIT
0048 3B4F1	0139	BNF SMRT ..THE SAME, UNDERFLOW NOT POSSIBLE
004A 521	0140	STR SP ..PUT IT BACK TO STACK
004B 9F1	0141	GHI AC ..OTHERWISE HAVE TO COMPARE SIGN OF RESULT
004C F31	0142	XOR ..SIGN BIT OF AC WAS STORED ON STACK
004D FF1	0143	SHL ..TAKE OUT THAT 7TH BIT
004E D51	0144	SMRT: SFP RETN ..DF=0 IF SUBTRACTION OK
004F :	0145	..
004F :	0146	..
004F :	0147	..16X16 BIT SIGNED MULTIPLY(2'S COMPLEMENT)
004F :	0148	..****AC=AC*OPRN
004F :	0149	..***** (TO CALL, WRITE)*****
004F :	0150	..****CALL MPYOP ; ,A(OPRN)
004F :	0151	..
004F 461	0152	MPYOP: LNA LINK ..FETCH MULTIPLICAND ADDRS
0050 BD1	0153	PHI MA ..INTO REGISTER A
0051 461	0154	LNA LINK
0052 AD1	0155	PLO MA ..FALL INTO MPY
0053 :	0156	.. 16X16 BIT SIGNED MULTIPLY (2'S COMPLEMENT)
0053 :	0157	.. CALL HERE IF OPERAND ADDRESS
0053 :	0158	.. IN REGISTER MA
0053 :	0159	.. ****AC=AC*M(R(MA))
0053 :	0160	.. ***** (TO CALL, WRITE) *****
0053 :	0161	.. ****CALL MPY
0053 :	0162	..
0053 F01	0163	MPY: SEX MA .. SET X NOW
0054 9F1	0164	GHI AC ..CHECK IS THE SIGN OF MULTICAND
0055 F31	0165	XOR ..THE SAME AS THE SIGN OF MULTIPR
0056 FA801	0166	ANI #80 ..AND STORE THAT BIT
0058 BC1	0167	PHI CR ..INTO CR.1
0059 FA101	0168	LDI #10 .. SET COUNTER TO 16

005R AC1	0169	PLO CR
005C F8001	0170	LDI #00 .. INITIALIZE MQ TO 0
005E BF1	0171	PHI MQ ... TO HOLD PRODUCT.
005F AF1	0172	PLO MQ
0060 2C1	0173 MPL:	DEC CR ..IF NOT, DECREMENT IT.
0061 9F1	0174	GHI AC .. SHIFT AC (=MULTIPLIER) RIGHT
0062 F61	0175	SHR
0063 RF1	0176	PHI AC
0064 RF1	0177	GLO AC
0065 761	0178	SHRC .. SHIFT 0 ACROSS BYTES ..
0066 AF1	0179	PLO AC
0067 9F1	0180	GHI MQ
006R 3R761	0181	RNF MPB ..IF NO BIT OUT, DON'T ADD.
006A 1D1	0182	INC MA .. POINT TO LOW 8 OF MULTIPLICAND
006R AC1	0183	GLO CR .. IF NOT LAST ITERATION,
006C 3A791	0184	RNZ MPA .. GO ADD.
006F RF1	0185	GLO MQ ..LOAD MQ.0 INTO D
006F F71	0186	SM ..MQ.0-(MA.0)
0070 AF1	0187	PLO MQ ..AND STORE RACK INTO MQ.0
0071 2D1	0188	DEC MA ..NOW DO THE HIGH BYTE
0072 9F1	0189	GHI MQ ..
0073 771	0190	SMB ..REMEMBER THE BORROW BIT
0074 RF1	0191	PHI MQ ..AND PUT RACK INTO MQ.1
0075 9C1	0192	GHI CR ..NOW ARE THE SIGNS OF OPERANDS THE SAME
0076 FF1	0193 MPB:	SHL ..TEST FOR SIGN BIT
0077 30R31	0194	RR MPS+#03 ..IF NEGATIVE, SIGN EXTEND
0079 RF1	0195 MPA:	GLO MQ ..DO MQ+(MA),
007A F41	0196	ADD ..MQ.0+(MA.0)
007R AF1	0197	PLO MQ ..AND PUT RACK INTO MQ.0
007C 2D1	0198	DEC MA ..SAME FOR HIGH BYTE
007D 9F1	0199	GHI MQ ..
007F 741	0200	ADC ..ADD WITH CARRY
007F RF1	0201	PHI MQ ..
0080 CF1	0202 MPS:	LSDF
0081 F01	0203	LDX
0082 FF1	0204	SHL..IF OPERAND IS NEGATIVE, THEN
0083 9F1	0205	GHI MQ ..PUT MQ.1 INTO D
0084 761	0206	SHRC .. SHIFT IN CARRY
0085 RF1	0207	PHI MQ
0086 RF1	0208	GLO MQ .. CONTINUE TO LOW 8 OF MQ
0087 761	0209	SHRC
0088 AF1	0210	PLO MQ
0089 3R8F1	0211	RNF MT .. IF NO CARRY OUT, ITERATE.
008R 9F1	0212	GHI AC .. ADD CARRY OUT INTO AC MSR.
008C F9R01	0213	ORI #80
008F BF1	0214	PHI AC
008F AC1	0215 MT:	GLO CR ..CHECK COUNTER
0090 3A601	0216	RNZ MPL ..IF COUNTER IS NOT 0, GO BACK FOR MORE
0092 9F1	0217 MPX:	GHI AC .. FINISHED;
0093 FF1	0218	SHL .. CHECK FOR PRODUCT > 15 BITS.
0094 RF1	0219	GLO MQ
0095 C71	0220	LSNF ..THAT'S HIGH 17 BITS
0096 FFFF1	0221	XRI #FF .. ALL 00 OR FF.
009R 3A9F1	0222	RNZ *+#06 .. NAW.
009A 9F1	0223	GHI MQ
009R C71	0224	LSNF
009C FFFF1	0225	XRI #FF
009F FFFF1	0226	ADI #FF .. SET DF IF PRODUCT > 15 BITS


```

00A0 D5:      0227      SEP RETN .. RETURN
00A1 :        0228      .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
00A1 :        0229      .. AC=MQ,AC/OPRN
00A1 :        0230      .. QUOTIENT IN AC , REMAINDER IN MQ
00A1 :        0231      .. ***** (TO CALL, WRITE) *****
00A1 :        0232      .. CALL DIVOP ; ,A(OPRN)
00A1 :        0233      ..
00A1 46:      0234      DIVOP: LDA LINK .. FETCH OPERAND ADDRESS
00A2 BD:      0235      PHI MA .. TO REGISTER MA
00A3 46:      0236      LDA LINK
00A4 AD:      0237      PLO MA .. FALL INTO DIVO/DIV/DIVQ.
00A5 :        0238      .. 32/16 BIT SIGNED DIVIDE (2'S COMPLEMENT)
00A5 :        0239      ... OPTION #1: CLEAR MQ AND CHECK FOR ZERO D
00A5 :        0240      .. ****AC=MQ,AC/M(R(MA))
00A5 :        0241      .. ***** (TO CALL, WRITE) *****
00A5 :        0242      .. *****CALL DIVO
00A5 FD:      0243      DIVO: SFX MA .. SET X TO POINT TO DIVISOR (0)
00A6 9F:      0244      GHI AC .. LOOK AT AC SIGN (0)
00A7 FF:      0245      SHL .. COPY IT TO DF (0)
00A8 F800:    0246      LDI #00 .. EXTEND #00 IF POSITIVE, (0)
00AA C7:      0247      LSNF .. (0)
00AB F8FF:    0248      LDI #FF .. #FF IF NEGATIVE. (0)
00AD BF:      0249      PHI MQ .. GIVING +0 OR -0 IN MQ (0)
00AF AF:      0250      PLO MQ .. (0)
00AF 4D:      0251      LDA MA .. CHECK FOR ZERO DIVISOR (0)
00B0 F1:      0252      OR .. (0)
00B1 2D:      0253      DFC MA .. (DON'T FORGET TO FIX POINTER)(0)
00B2 FD00:    0254      SDI #00 .. IF ZERO, CALL IT DIVIDE CHECK(0)
00B4 3AB7:    0255      RNF DIV ..GO ON IF NO DIVIDE CHECK ERR
00B6 D5:      0256      SEP RETN ..AND RETURN WITH DF=1 (0)
00B7 :        0257      ... OPTION #2: PERMIT 32-BIT DIVIDEND (/)
00B7 :        0258      ..
00B7 :        0259      ..MAKE SURE QUOTIENT DOES NOT EXCEED 16 BITS(/)
00B7 :        0260      .. ***** (TO CALL, WRITE) *****(/)
00B7 :        0261      .. ***** CALL DIV *****(/)
00B7 9F:      0262      DIV: GHI MQ ..SAVE PARTIAL DIVIDEND(/)
00B8 BC:      0263      PHI CR ..INTO CR.1(/)
00B9 BF:      0264      GLO MQ ..(/)
00BA AC:      0265      PLO CR ..AND CR.0(/)
00BB FF:      0266      SHL ..(/)
00BC AF:      0267      PLO MQ ..(/)
00BD 9F:      0268      GHI MQ ..DO THE SAME FOR HIGH BYTE(/)
00BF 7F:      0269      SHLC ..REMEMBER CARRY(/)
00BF BF:      0270      PHI MQ ..(/)
00C0 9F:      0271      GHI AC ..ALSO SHIFT IN AC HIGH(/)
00C1 FF:      0272      SHL ..(/)
00C2 33CD:    0273      RDE D2-#01 ..(/)
00C4 BC:      0274      GLO CR ..SEE IF MQ.0 =0(/)
00C5 3ACF:    0275      RNZ D2 ..IF NOT GO THROUGH CHECKING STEPS(/)
00C7 9C:      0276      GHI CR ..SEE IF MQ.1 IS 340(/)
00C8 FB40:    0277      XRI #40 ..WHICH SHOULD RESULT 0 IF TRUE(/)
00CA 32D7:    0278      BZ D4 ..IF TRUE,SKIP NORMAL CHECKING(/)
00CC 3B:      0279      ,#3B
00CD 1F:      0280      INC MQ ..IF 1, SHIFT INTO MQ(/)
00CF 9C:      0281      D2: GHI CR ..CHECK IF HIGH 2 BITS OF MQ ARE THE SAME(/)
00CF FACC:    0282      ANI #C0 ..TAKE OUT 2 HIGH BITS(/)
00D1 FD00:    0283      SDI #00 ..SEE IF THEY ARE THE SAME(/)
00D3 FF:      0284      SHL ..SHIFT OUT COMPARISON (/)
00D4 C30156: 0285      LRDF DVXX ..SET DF AND RETURN(/)

```

```

0007 9C1      0286 D4:  GHI CR  ..LOOK AT THE SIGNS OF DIVND AND DIVSR(/)
0008 ED1      0287  SEX MA  ..POINT X TO DIVISOR(/)
0009 F31      0288  XOR    ..(/)
000A FE1      0289  SHL    ..SET DF IF THE SAME(/)
000B 1D1      0290  INC MA  ..MQ.0-(MA)(/)
000C 8F1      0291  GLO MQ  ..MQ.0 TO D(/)
000D 33F61    0292  RDF DVA  ..IF FLAG ,MQ.0+(MA)(/)
000E F71      0293  SM    ..MQ.0-(MA.0)(/)
000F AF1      0294  PLO MQ  ..AND STORE BACK INTO MQ.0(/)
0010 2D1      0295  DEC MA  ..DO THE SAME FOR HIGH BYTE(/)
0011 9F1      0296  GHI MQ  ..LOAD MQ.1 INTO D(/)
0012 771      0297  SMR    ..MQ.1-(MA.1) WITH CARRY(/)
0013 30F81    0298  RR DVR  ..SKIP OVER ADD STEP(/)
0014 F41      0299 DVA:  ADD    ..MQ+(MA)(/)
0015 AF1      0300  PLO MQ  ..STORE BACK TO MQ.0(/)
0016 2D1      0301  DEC MA  ..SAME FOR HIGH BYTE(/)
0017 9F1      0302  GHI MQ  ..MQ.1+(MA.1)(/)
0018 741      0303  ADC    ..ADD MQ.1 AND (MA.1) WITH CARRY(/)
0019 BF1      0304 DVR:  PHI MQ  ..AND STORE BACK INTO MQ.1(/)
001A 9C1      0305  GHI CR  ..LOOK AT THE SIGN OF DIVND(/)
001B FF1      0306  SHL    ..SET DF IF DIVND IS NEGATIVE
001C C80FA1   0307  LBNF DV2  ..IF NOT, DON'T COMPLEMENT DIFF(/)
001D 9E1      0308  GHI MQ  ..COMPLEMENT MQ(/)
001E FFFF1    0309  XRI #FF  ..BY XOR TO #FF(/)
001F BF1      0310  PHI MQ  ..(/)
0020 8F1      0311  GLO MQ  ..(DO THE SAME FOR LOW BYTE)(/)
0021 FFFF1    0312  XRI #FF  ..(/)
0022 AF1      0313  PLO MQ  ..(/)
0023 1F1      0314  INC MQ  ..REMEMBER TO ADD 1 (/)
0024 9E1      0315 DV2:  GHI MQ  ..NOW LOOK AT THE DIFF OF MQ&(MA)(/)
0025 C201031  0316  LBZ D10  ..IF 0,CHECK MQ=0(/)
0026 FF1      0317  SHL    ..CHECK IF MQ IS NEGATIVE(/)
0027 335R1    0318  RDF DDQ  ..IF YES, NO PROBLEM(/)
0028 30541    0319  RR D9   ..IF NOT 0 NOR NEG,DIVND IS TOO LARGE (/)
0029 8F1      0320 D10:  GLO MQ  ..HERE WE CHECK DIFF=0 CASES(/)
002A FFFF1    0321  ADI #FE  ..IF MQ.0 IS NOT EITHER 0 OR 1(/)
002B 33561    0322  RDF DVXX  ..THEN DIVND IS STILL TOO LARGE(/)
002C FFFF1    0323  XRI #FF  ..RESULT 0 IF MQ.0 WAS 1(/)
002D 3A391    0324  RNZ DVH  ..IF NOT, MUST BE 0, GO TO DVH(/)
002E 9C1      0325  GHI CR  ..SEE IF DIVND IS NEGATIVE(/)
002F FA801    0326  ANI #80  ..RESULT 80 IF YES(/)
0030 32541    0327  RZ    DO    ..IF DIVND POSITIVE, IT CANNOT DIVIDE(/)
0031 F41      0328  ADD    ..SEE IF DIVISOR IS POSITIVE(/)
0032 33561    0329  RDF DVXX  ..IF NEGATIVE, DIVND CANNOT DIVIDE(/)
0033 1D1      0330  INC MA  ..AC+(MA) TO MQ(/)
0034 8F1      0331  GLO AC  ..DO AC LOW FIRST(/)
0035 F41      0332  ADD    ..(/)
0036 AF1      0333  PLO MQ  ..STORE BACK TO LOW MQ(/)
0037 2D1      0334  DEC MA  ..DO THE SAME FOR HIGH BYTE(/)
0038 9F1      0335  GHI AC  ..(/)
0039 741      0336  ADC    ..ADD WITH CARRY(/)
003A BF1      0337  PHI MQ  ..(/)
003B 1D1      0338  INC MA  ..LEAVE MA POINTING TO LOW DIVISOR(/)
003C F01      0339  LOX    ..NOW CHECK LOW BIT (MA) IS 0 OR 1(/)
003D F61      0340  SHR    ..SHIFT THAT LOW BIT OUT(/)

```

011F 2D1	0341	DEC MA .. REMEMBER TO RESET MA(//)
0120 9E1	0342	GHI MQ ..READY TO ADD #80(//)
0121 C71	0343	LSNF ..IF LOW BIT OF AC IS 1(//)
0122 FCR01	0344	ADI #80 ..TO MQ.1(//)
0124 RE1	0345	PHI MQ ..STORE STATUS(//)
0125 FF1	0346	SHL ..SEE IF MQ >0(//)
0126 9C1	0347	GHI CR ..SEE IF DIVND POSTIVE(//)
0127 FAR01	0348	ANI #80 ..RY TAKE OUT SIGN(//)
0129 3A2F1	0349	RNZ *+H06 ..(//)
012R 3B541	0350	RNF D9 ..EXIT WITH DF=1(//)
012D 305R1	0351	RR DDQ ..OK(//)
012F 33561	0352	RDF DVXX ..IF NEGATIVE, THEN OUT(//)
0131 9F1	0353	GHI MQ ..SEE IF MQ=0?(//)
0132 3A5R1	0354	RNZ DDQ ..IF NOT, THEN NO PROBLEM(//)
0134 RE1	0355	GLO MQ ..MAKE SURE MQ.0 IS 0 TOO(//)
0135 32541	0356	RZ D9 ..IF YES, THEN RETURN WITH DF=1(//)
0137 305R1	0357	RR DDQ ..OR ELSE GO TO DIVIDE(//)
0139 9C1	0358 DVH:	GHI CR ..SEE IF THE OPERANDS SIGNS DIFF(//)
013A F31	0359	XOR ..BY COMPARING SIGN BITS(//)
013R FF1	0360	SHL ..AND TAKE OUT THAT BIT(//)
013C 33141	0361	RDF D10+H11 ..IF SINGS DIFF, IT'S OK(//)
013F 9C1	0362	GHI CR ..OTHERWISE TEST SIGN OF DIVND(//)
013F FF1	0363	SHL ..IF POSITIVE, RETURN WITH DF=1(//)
0140 3B541	0364	RNF D9 ..RETURN WITH DF=1(//)
0142 1D1	0365	INC MA ..SEE IF LOW BIT OF AC IS 0 OR 1(//)
0143 F01	0366	LDX ..LOAD THAT IN D(//)
0144 2D1	0367	DFC MA ..REMEMBER TO RESET MA(//)
0145 F61	0368	SHR ..SHIFT THAT BIT OUT(//)
0146 RF1	0369	GLO AC ..IF AC IS NOT 0, NOT PROBLEM(//)
0147 3A5R1	0370	RNZ DDQ ..GO TO DIVIDE(//)
0149 9F1	0371	GHI AC ..READY TO CHECK #80 IF LOW BIT AC IS 1(//)
014A 33501	0372	RDF *+H06 ..(//)
014C 32541	0373	RZ D9 ..NO GOOD, RETURN WITH DF=1(//)
014F 305R1	0374	RR DDQ ..ANY THING ELSE IS OK(//)
0150 FRR01	0375	XRI #80 ..IF AC.1 IS'NT #80(//)
0152 3A5R1	0376	RNZ DDQ ..IT'S OK(//)
0154 FF001	0377 D9:	SMI #00 ..DF IS SET TO 1(//)
0156 9C1	0378 DVXX:	GHI CR ..PUT ORIGINAL DIVND(//)
0157 RF1	0379	FHI MQ ..INTO MQ(//)
0158 AC1	0380	GLO CR ..(//)
0159 AF1	0381	PLO MQ ..(//)
015A D51	0382	SFP RETN ..AND RETURN WITH DF=1(//)
015R 9C1	0383 DDQ:	GHI CR ...PUT ORIGINAL DIVND(//)
015C RF1	0384	PHI MQ ..BACK INTO MQ(//)
015D AC1	0385	GLO CR ..(//)
015E AF1	0386	PLO MQ ..(//)
015F 1	0387	... OPTION #3: ASSUME BENIGN PROGRAM; NOCHECK
015F 1	0388	.. ***** (TO CALL, WRITE) *****
015F 1	0389	.. *****CALL DIVQ
015F 9F1	0390 DIVQ:	GHI MQ ..LOOK AT DIVIDEND SIGN
0160 FF1	0391	SHL .. IF POSITIVE,
0161 FR901	0392	LDI #90 .. PLAN TO BEGIN WITH SUBTRACT,
0163 C71	0393	LSNF .. (ALSO SAVE SIGN OF DIVIDEND)
0164 FR501	0394	LDI #50 .. OTHER WISE BEGIN WITH ADD.
0166 AC1	0395	PLO CR .. SET ITERATION COUNT IN CR.0
0167 RF1	0396 DVL:	GLO MQ ..SHIFT MQ LEFT 1 BIT
0168 FF1	0397	SHL ..SHIFT LPFT MQ.0
0169 AF1	0398	PLO MQ ..
016A 9F1	0399	GHI MQ ..SAME FOR HIGH BYTE

016B 7E1	0400	SHLC .. SHIFT LEFT WITH CARRY
016C RE1	0401	PHI MQ ..
016D AF1	0402	GLO AC ..SHIFT AC LEFT 1 BIT
016E FE1	0403	SHL ..SHIFT AC.0
016F AF1	0404	PLO AC ..
0170 9F1	0405	GHI AC ..
0171 7F1	0406	SHLC ..SHIFT WITH CARRY
0172 RF1	0407	PHI AC
0173 3R761	0408	BNF *+H03 .. BIT SHIFTED OUT OF AC.1,
0175 1F1	0409	INC MQ .. GOES INTO MQ.0
0176 FD1	0410	SEX MA ..
0177 RC1	0411	GLO CR .. NOW, WAS THAT ADD, OR SUBTRACT?
017A F31	0412	XOR .. IT DEPENDS ON SAVED FLAG,
0179 FF1	0413	SHL ..AND SIGN OF DIVISOR
017A 1D1	0414	INC MA ..MQ.0-(MA.0)
017B AF1	0415	GLO MQ ..
017C 3R851	0416	RNF DSA ..IF NO FLAG, MQ.0+(MA.0)
017F F71	0417	SM ..
017F AF1	0418	PLO MQ ..
0180 2D1	0419	DEC MA ..FIX X PTR
0181 9F1	0420	GHI MQ ..DO THE SAME FOR HIGH BYTE
0182 771	0421	SMR ..REMEMBER THAT BORROW BIT
0183 30AA1	0422	RR DSM ..SKIP OVER ADD STEPS
0185 F41	0423 DSA:	ADD ..MQ+(MA)
0186 AF1	0424	PLO MQ ..
0187 2D1	0425	DEC MA ..DO THE SAME FOR HIGH BYTE
0188 9F1	0426	GHI MQ ..
0189 741	0427	ADC ..ADD WITH CARRY
018A RF1	0428 DSM:	PHI MQ ..STORE BACK TO MQ.1
018B 2C1	0429	DFC CR .. COUNT DOWN ITERATION COUNTER
018C RC1	0430	GLO CR
018D FA7F1	0431	ANI #7F
018F 3B941	0432	BNF *+H05 .. TEST CARRY OUT OF ADD/SUBTRACT
0191 1F1	0433	INC AC .. IF 1, SHIFT INTO QUOTIENT,
0192 F9801	0434	ORI #80 .. AND FLAG NEXT OP AS SUBTRACT.
0194 AC1	0435	PLO CR .. OTHERWISE IT'S ADD.
0195 FA3F1	0436	ANI #3F .. LOOK AT COUNTER:
0197 3A671	0437	RNZ DVL .. IF NOT 0, LOOP BACK;
0199 33R01	0438	RDF DVR ...AT END, CHECK REMAINDER ADJUST.
019B 381	0439	,#38
019C 1F1	0440	DVC: INC AC .. (FINAL DIVIDE STEP)
019D F31	0441	XOR .. BE SURE TO GET POLARITY
019F FF1	0442	SHL .. OF ADJUSTMENT RIGHT...
019F 1D1	0443	INC MA .. YES, ADD DIVISOR BACK ON,
01A0 RE1	0444	GLO MQ .. TO CORRECT FOR FINAL SUBTRACT,
01A1 33AA1	0445	RDF DVM .. (ADDING NEGATIVE IS SURT.)
01A3 F41	0446	ADD .. WHICH SHOULDN'T HAVE.
01A4 AF1	0447	PLO MQ
01A5 2D1	0448	DEC MA
01A6 9F1	0449	GHI MQ
01A7 741	0450	ADC ..ADD WITH CARRY
01A8 30AF1	0451	RR DVR-#01
01AA F71	0452	DVM: SM .. SAME THING,
01AB AF1	0453	PLO MQ .. EXCEPT, FOR NEGATIVE DIVISOR.
01AC 2D1	0454	DEC MA
01AD 9E1	0455	GHI MQ
01AF 771	0456	SMR
01AF RF1	0457	PHI MQ
01B0 9F1	0458	DVR: GHI MQ .. IF REMAINDER IS NOT ZERO,
01B1 3AB61	0459	RNZ *+H05

01B3 8E1	0460	GLO MQ
01B4 32C11	0461	RZ DVN .. PUT IT IS; NO PROBLEM.
01B6 8C1	0462	GLO CR .. IF NOT ZERO,
01B7 FF1	0463	SHL .. IT SHOULD BE SAME SIGN
01B8 FF1	0464	SHL .. AS ORIGINAL DIVIDEND.
01B9 9E1	0465	GHI MQ
01BA CF1	0466	LSDF
01BB F8B01	0467	XRI #80
01BD FC801	0468	ADI #80 .. IF NOT, WE NEED
01BF 3B9C1	0469	RNF DVC .. ONE MORE DIVIDE ITERATION.
01C1 F01	0470	DVN: LDX .. FINALLY, IF DIVISOR NEGATIVE,
01C2 FF1	0471	SHL
01C3 3BD01	0472	RNF DVX .. (IT'S NOT; WE ARE DONE)
01C5 AF1	0473	GLO AC .. COMPLEMENT QUOTIENT,
01C6 FBFF1	0474	XRI #FF .. BY INVERTING IT,
01C8 AF1	0475	PLO AC
01C9 9F1	0476	GHI AC
01CA FBFF1	0477	XRI #FF
01CC BF1	0478	PHI AC
01CD 1F1	0479	INC AC .. THEN INCREMENTING
01CE FC001	0480	ADI #00 .. ALSO CLEAR DF.
01D0 D51	0481	DVX: SEP RETN .. DF=0 IF DIVIDE SUCCESSFUL.
01D1 1	0482	..
01D1 1	0483	.. 16-BIT ADD TO AC, OPERAND ADDRESS IN CALL
01D1 1	0484	.. ****AC=AC+OPRN .. OPRN=2 BYTE OPERAND
01D1 1	0485	.. ***** (TO CALL, WRITE) *****
01D1 1	0486	.. ****CALL ADDOP ; ,A(OPRN)'
01D1 1	0487	..
01D1 1	0488	..
01D1 461	0489	ADDOP: LDA LINK .. FETCH OPERAND ADDRESS
01D2 BD1	0490	PHI MA .. TO REGISTER MA
01D3 461	0491	LDA LINK
01D4 AD1	0492	PLO MA .. FALL INTO ADD
01D5 1	0493	.. 16-BIT ADD TO AC, OPERAND ADDRESS IN REGI
01D5 1	0494	.. CALL HERE IF OPRN ADDRESS
01D5 1	0495	.. IS IN REGISTER MA
01D5 1	0496	.. ****AC=AC+M(R(MA))
01D5 1	0497	.. ***** (TO CALL, WRITE) *****
01D5 1	0498	.. ****CALL ADD
01D5 1	0499	..
01D5 FD1	0500	ADD: SFX MA ..CHECK SIGN BIT OF AC
01D6 9F1	0501	GHI AC ..GET THE OPERAND
01D7 F31	0502	XOR ..AND OPERAND QMA
01D8 F8B01	0503	XRI #80 ..RESULT A 1 IF DIFF
01DA 521	0504	STR SP ..STORE ON STACK
01DB 1D1	0505	INC MA ..POINT TO LOW 8 BITS
01DC 8F1	0506	GLO AC .. FETCH AC LOW 8
01DD F41	0507	ADD .. ADD LOW 8 FROM MEMORY
01DE AF1	0508	PLO AC .. PUT IT BACK
01DE 2D1	0509	DEC MA .. POINT TO HIGH 8 MEMORY LOCATION
01E0 9F1	0510	GHI AC
01E1 741	0511	ADC ..ADD HIGH BYTE WITH CARRY
01E2 BF1	0512	PHI AC .. PUT IN AC.
01E3 021	0513	LON SP ..LOAD THE STORED COMPARING SIGN BIT
01E4 FF1	0514	SHL ..RESET STACK PTR
01E5 3BFA1	0515	RNF ADDRT+#01 ..NOT POSSIBLE
01E7 9F1	0516	GHI AC ..OTHERWISE SEE IF SUM IS RIGHT
01E8 F31	0517	XOR ..BY COMPARING SIGN BITS
01E9 FF1	0518	ADDRT: SHL ..SHIFT OUT 0 INTO DF
01FA D51	0519	SEP RETN ..RETURN TO MAIN

```

01ER :      0520      ..
01ER :      0521      .. 16-BIT CONSTANT ADD TO AC.
01ER :      0522      .. CALL HERE FOR ADD CONSTANT TO AC
01ER :      0523      .. ****AC=AC+CONSTANT
01ER :      0524      .. ***** (TO CALL, WRITE) *****
01FR :      0525      .. ****CALL ADDCON ; ,CONSTANT
01FR :      0526      ..
01FR R6:      0527      ADDCON: GLO LINK .. COPY LINK TO MA
01FC AD:      0528      PLO MA
01FD 96:      0529      GHI LINK
01FF RD:      0530      PHI MA
01FF 16:      0531      INC LINK .. INCREMENT PAST DATUM
01F0 16:      0532      INC LINK
01F1 30D5:    0533      RR ADD .. GO ADD.
01F3 :      0534      ..
01F3 :      0535      .. 16-BIT ADD FROM TOP OF STACK
01F3 92:      0536      ADDST: GHI SP .. COPY STACK POINTER
01F4 RD:      0537      PHI MA .. TO MA REGISTER
01F5 A2:      0538      GLO SP
01F6 AD:      0539      PLO MA
01F7 1D:      0540      INC MA .. ADVANCE TO SUB-TOP (Q)
01F8 1D:      0541      INC MA .. (Q)
01F9 1D:      0542      INC MA .. (Q)
01FA 30D5:    0543      RR ADD .. GO DO IT
01FC :      0544      ..
01FC :      0545      ..
01FC :      0546      ..
01FC :      0547      .. PUSH AC INTO STACK
01FC :      0548      .. STACK POINTER = SP
01FC :      0549      .. ***** (TO CALL, WRITE) *****
01FC :      0550      .. ****CALL PUSHAC
01FC :      0551      .. PUSH AC (UNDER TOP OF STACK)
01FC 92:      0552      PUSHAC: GHI SP .. COPY STACK POINTER TO MA
01FD RD:      0553      PHI MA
01FF A2:      0554      GLO SP
01FF AD:      0555      PLO MA
0200 1D:      0556      INC MA .. NOW SLICE OFF TOP 2 BYTES,
0201 1D:      0557      INC MA
0202 F2:      0558      SEX SP
0203 0D:      0559      LDN MA .. TO MAKE A 2-BYTE HOLE.
0204 73:      0560      STXD
0205 2D:      0561      DEC MA
0206 0D:      0562      LDN MA
0207 73:      0563      STXD
0208 9F:      0564      GHI AC .. NOW STUFF AC INTO THE HOLE.
0209 5D:      0565      STR MA
020A 8F:      0566      GLO AC
020B 1D:      0567      INC MA
020C 5D:      0568      STR MA
020D 2D:      0569      DEC MA .. LEAVE MA POINTING TO IT.
020F 05:      0570      SEP RETN .. (AC UNCHANGED)
020F :      0571      .. PUSH CR, MA, MQ (UNDER TOP OF STACK)
020F RD:      0572      PUSHCR: GLO MA .. FIRST PUSH MA ONTO TOP
0210 F2:      0573      SEX SP
0211 73:      0574      STXD
0212 9D:      0575      GHI MA
0213 52:      0576      STR SP
0214 92:      0577      GHI SP .. NOW COPY SP TO MA
0215 RD:      0578      PHI MA

```


0216 R2:	0579	GLO SP
0217 AD:	0580	PLO MA
0218 22:	0581	DFC SP
0219 1D:	0582	INC MA .. THEN ADJUST IT
021A 1D:	0583	INC MA
021B 1D:	0584	INC MA .. TO POINT INTO OLD TOP
021C RF:	0585	GLO MQ .. CONTINUE PUSHING, MQ
021D 73:	0586	STXD
021F 9F:	0587	GHI MQ
021F 73:	0588	STXD
0220 0D:	0589	LDN MA
0221 73:	0590	STXD
0222 2D:	0591	DEC MA
0223 0D:	0592	LDN MA
0224 73:	0593	STXD
0225 9C:	0594	GHI CR .. FINALLY INSERT CR IN HOLF
0226 5D:	0595	STR MA
0227 1D:	0596	INC MA
0228 8C:	0597	GLO CR
0229 5D:	0598	STR MA
022A 05:	0599	SEP RETN .. (MA IS GARBAGE OUT)
022B :	0600	..
022B :	0601	.. POP STACK INTO MQ, MA, CR (UNDER TOP OF S
022B 12:	0602	POPQ: INC SP
022C 92:	0603	GHI SP .. COPY STACK POINTER TO MA
022C RD:	0604	PHI MA
022F 82:	0605	GLO SP
022F AD:	0606	PLO MA
0230 1D:	0607	INC MA .. ADJUST POINTER TO CR DATUM
0231 1D:	0608	INC MA
0232 1D:	0609	INC MA
0233 1D:	0610	INC MA
0234 1D:	0611	INC MA
0235 1D:	0612	INC MA
0236 4D:	0613	LDA MA .. FFTCH IT
0237 RC:	0614	PHI CR
0238 0D:	0615	LDN MA
0239 AC:	0616	PLO CR
023A 2D:	0617	DEC MA .. COPY TOP OF STACK INTO GAP
023B 42:	0618	LDA SP
023C 5D:	0619	STR MA
023D 1D:	0620	INC MA
023F 42:	0621	LDA SP
023F 5D:	0622	STR MA
0240 42:	0623	LDA SP .. THEN POP MQ
0241 RF:	0624	PHI MQ
0242 42:	0625	LDA SP
0243 AF:	0626	PLO MQ
0244 42:	0627	LDA SP .. FINALLY POP MA
0245 RD:	0628	PHI MA
0246 02:	0629	LDN SP
0247 AD:	0630	PLO MA
0248 05:	0631	SEP RETN
0249 :	0632	.. 16-BIT ACCUMULATOR LOAD (ADDRESS IN CALL)
0249 :	0633	.. ****AC=OPRN
0249 :	0634	.. ***** (TO CALL, WRITE) *****
0249 :	0635	.. ****CALL LOADOP ; ,A(OPRN)
0249 :	0636	..
0249 46:	0637	LOADOP: LDA LINK .. FFTCH ADDRESS
024A RD:	0638	PHI MA .. TO MA REGISTER

```

024R 46:      0639      LDA LINK
024C AD:      0640      PLO MA .. FALL INTO LOAD
024D :      0641      .. 16-BIT ACCUMULATOR LOAD (ADDRESS IN MA)
024D :      0642      .. CALL HERE IF OPERAND
024D :      0643      .. ADDRESS IN REGISTER MA
024D :      0644      .. ****AC=M(R(MA))
024D :      0645      .. ***** (TO CALL, WRITE) *****
024D :      0646      .. ****CALL LOAD
024D :      0647      ..
024D 4D:      0648      LOAD: LDA MA .. FFTCH HIGH 8
024E RF:      0649      PHI AC
024F 4D:      0650      LDA MA .. NOW LOW 8
0250 AF:      0651      PLO AC .. LEAVE MA AT NEXT DOUBLE-BYTE
0251 D5:      0652      SEP RETN .. GFF, THAT WAS QUICK.
0252 :      0653      ..
0252 :      0654      ..
0252 :      0655      .. 16-BIT ACCUMULATOR LOAD FROM CONSTANT IN
0252 :      0656      .. ****AC=CONSTANT
0252 :      0657      .. ***** (TO CALL, WRITE) *****
0252 :      0658      .. ****CALL LODCON ; ,CONSTANT
0252 :      0659      ..
0252 46:      0660      LODCON: LDA LINK .. FETCH HIGH 8
0253 RF:      0661      PHI AC
0254 46:      0662      LDA LINK .. THEN LOW 8
0255 AF:      0663      PLO AC
0256 D5:      0664      SEP RETN
0257 :      0665      ..
0257 :      0666      .. 16-BIT ACCUMULATOR STORE (ADDRESS IN CALL
0257 :      0667      .. ****OPRN=AC
0257 :      0668      .. ***** (TO CALL, WRITE) *****
0257 :      0669      .. ****CALL STOROP ; ,A(OPRN)
0257 :      0670      ..
0257 46:      0671      STOROP: LDA LINK .. FETCH ADDRESS INTO MA
0258 RD:      0672      PHI MA
0259 46:      0673      LDA LINK
025A AD:      0674      PLO MA .. THEN FALL INTO STORE
025B :      0675      .. 16-BIT ACCUMULATOR STORE (ADDRESS IN MA)
025B :      0676      .. ****M(R(MA))=AC
025B :      0677      .. ***** (TO CALL, WRITE) *****
025B :      0678      .. ****CALL STORE
025B :      0679      ..
025B 9F:      0680      STORE: PHI AC .. FIRST HIGH 8
025C 5D:      0681      STR MA
025D 1D:      0682      INC MA .. INCREMENT MA, SINCE STR DOESN'T
025E AF:      0683      GLO AC .. NOW LOW 8
025F 5D:      0684      STR MA
0260 1D:      0685      INC MA .. LEAVE MA POINTING TO NEXT WORD
0261 D5:      0686      SEP RETN .. QUIT.
0262 :      0687      ..
0262 :      0688      ..
0262 :      0689      .. 16-BIT COMPARE, OPERAND ADDRESS IN CALL
0262 :      0690      .. ****AC-OPRN (DF SET IF 0 OR +)
0262 :      0691      .. ***** (TO CALL, WRITE) *****
0262 :      0692      .. ****CALL COMOP ; ,A(OPRN)
0262 :      0693      ..
0262 46:      0694      COMOP: LDA LINK .. FETCH ADDRESS
0263 RD:      0695      PHI MA .. TO MA REGISTER
0264 46:      0696      LDA LINK
0265 AD:      0697      PLO MA
0266 :      0698      .. 16-BIT COMPARE, OPERAND ADDRESS IN REGIST

```

```

0266 : 0699 .. CALL HERE IF OPERAND
0266 : 0700 .. ADDRESS IN REGISTER MA
0266 : 0701 .. ****AC-M(R(MA)) (DF SET IF 0 OR +)
0266 : 0702 .. ***** (TO CALL, WRITE) *****
0266 : 0703 .. ****CALL COMP
0266 : 0704 ..
0266 FD: 0705 COMP: SEX MA .. COMPARE HIGH 8 FRST
0267 9F: 0706 GHI AC ..CHECK IF SIGN OF OPERANDS ARE SAME
0268 F3: 0707 XOR ..BY LOOKING AT THE HIGHEST SIGN BIT
0269 FA80: 0708 ANI #80 ..RESULT A '1' IF NEGATIVE
026B 3A76: 0709 BNZ CNE ..IF NOT, THEN GO TO CNE
026D 9F: 0710 GHI AC
026F F7: 0711 SM
026F 3A77: 0712 RNZ CMPX .. NOT EQUAL QUIT
0271 1D: 0713 INC MA .. TRY LOW 8
0272 AF: 0714 GLO AC
0273 F7: 0715 SM
0274 2D: 0716 DEC MA .. LEAVE MA POINTING TO IT
0275 3A: 0717 ,#3A
0276 F4: 0718 CNE: ADD ..SEE IF OPERAND IS NEGATIVE
0277 D5: 0719 CMPX: SEP RETN ..DF=1 IF AC>=(MA)
0278 : 0720 ..
0278 : 0721 ..
0278 : 0722 ..
0278 : 0723 ..
0278 : 0724 ..TEST 16-BIT ACCUMULATOR SIGN/ZERO .....
027A 16: 0725 INC LINK ..SKIP OVER NON ZERO RETURN
027A 16: 0726 INC LINK ..
027A D5: 0727 SEP RETN
027B 9F: 0728 TEST: GHI AC .. FIRST LOOK AT SIGN
027C FF: 0729 SHL ..SET DF IF MINUS
027D 9F: 0730 GHI AC .. NOW CHECK FOR 0
027F 3A7A: 0731 RNZ TEST-#03 .. NO.
0280 AF: 0732 GLO AC
0281 327A: 0733 RZ TEST-#01 .. GO TAKE ZERO EXIT.
0283 307A: 0734 RR TEST-#03 .. GO TAKE NON-ZERO EXIT.
0285 : 0735 .. POP STACK (INTO AC) (UNDER TOP)
0285 : 0736 .. POPS 2 BYTES FROM STACK INTO AC
0285 : 0737 .. ***** (TO CALL, WRITE) *****
0285 : 0738 .. *****CALL POPAC
0285 FF00: 0739 POPAC: SMI #00 .. SFT DF TO REMEMBER THIS ENTRY
0287 CA: 0740 LSKP
0288 FC00: 0741 POP: ADI #00 .. CLEAR DF FOR THIS ENTRY
028A 12: 0742 INC SP
028B 92: 0743 GHI SP .. COPY SP TO MA
028C BD: 0744 PHI MA
028D A2: 0745 GLO SP
028F AD: 0746 PLO MA
02AF 1D: 0747 INC MA .. ADJUST TO SUB-TOP OF STACK
0290 1D: 0748 INC MA
0291 3B98: 0749 BNE *+#07
0293 4D: 0750 LDA MA .. POPPING INTO AC. GET DATUM
0294 AF: 0751 PHI AC
0295 0D: 0752 LDN MA
0296 AF: 0753 PLO AC
0297 2D: 0754 DEC MA
0298 42: 0755 LDA SP .. NOW CLOSE UP THE GAP
0299 5D: 0756 STR MA
029A 1D: 0757 INC MA
029B 02: 0758 LDN SP

```


029C 5D1	0759	STR MA
029D 1D1	0760	INC MA
029E D51	0761	SEP RETN .. MA POINTS TO NEW SUB-TOP
029F :	0762	..
029F :	0763	.. DECIMAL TO BINARY CONVERSION
029F :	0764	.. ***** AC=DECIMAL NUMBER OF N BYTES
029F :	0765	.. DECIMAL NUMBER = SIGN,NN,.....,N1,N0
029F :	0766	.. SIGN=#0B +
029F :	0767	.. SIGN=#0D -
029F :	0768	.. N0=10**0 DIGIT
029F :	0769	.. N1=10**1 DIGIT
029F :	0770	.. ***** (TO CALL, WRITE) *****
029F :	0771	.. ***** CALL CDB: ,A(NUMBER); ,LENGTH
029F 461	0772 CDB:	LDA LINK ..GET NUM ADDRESS
02A0 RA1	0773	PHI AR ..AND STORE IN RA.
02A1 461	0774	LDA LINK
02A2 AA1	0775	PLO AR
02A3 461	0776	LDA LINK ..GET LENGTH
02A4 FF011	0777	SMI #01 ..MINUS SIGN BYTE
02A6 AR1	0778	PLO NR ..AND STORE IN NR.
02A7 FB001	0779	LDI #00 ..CLEAR RF
02A9 AF1	0780	PLO AC
02AA RF1	0781	PHI AC
02AB 0A1	0782	LDN AR ..CHECK SIGN BYTE
02AC FB0D1	0783	XRI #0D ..MINUS?
02AF BR1	0784	PHI NR ..INTO NR.1
02AF 1A1	0785 LOOP:	INC AR ..GRAB THE FIRST DIGIT
02B0 E21	0786	SEX SP ..FIX X PNTR
02B1 0A1	0787	LDN AR ..CLEAR HIGH BYTE (FOR ASCII)
02B2 FA0F1	0788	ANI #0F
02B4 521	0789	STR SP ..PUT IT BACK
02B5 BF1	0790	GLO AC ..ADD THAT DIGIT TO ACCUM
02B6 F41	0791	ADD
02B7 AF1	0792	PLO AC
02B8 9F1	0793	GHI AC
02B9 7C001	0794	ADCI #00 ..REMEMBER CARRY OVER
02BA RF1	0795	PHI AC
02BC 33EF1	0796	RDF OVFLW ..EXCEEDS ACCUM LIMIT?
02BF 2B1	0797	DEC NR ..DEC DIGIT COUNTER
02BF AB1	0798	GLO NR ..SEE IF IT IS 0?
02C0 32F01	0799	RZ FINAL ..YES, THEN DONE
02C2 BF1	0800	GLO AC ..OTHERWISE MULTIPLY THE ACC BY 10
02C3 FF1	0801	SHL
02C4 731	0802	STXD
02C5 9F1	0803	GHI AC
02C6 7F1	0804	SHLC ..CARRY OVER
02C7 731	0805	STXD
02C8 33FD1	0806	RDF OVFLW-#02 ..EXCEEDED ACC LIMIT
02CA FB021	0807	LDI #02
02CC 521	0808 MPY3:	STR SP ..LOOP COUNT
02CD BF1	0809	GLO AC
02CF FE1	0810	SHL ..NOW SHIFT AC OVER 4 TIMES MORE
02CF AF1	0811	PLO AC
02D0 9F1	0812	GHI AC ..SAME FOR AC.1
02D1 7F1	0813	SHLC
02D2 BF1	0814	PHI AC
02D3 33ED1	0815	RDF OVFLW-#02 ..IF OVERFLOW, RESFT STACK
02D5 021	0816	LDN SP ..CHECK LOOP COUNT
02D6 32DC1	0817	RZ MPY10 ..AFTER MULPLY BY 8
02DA FF011	0818	SMI #01 ..OR ELSE DEC LOOP COUNT

```

02DA 30CC:      0819      RR MPY3  ..BACK FOR MORE ADDITION
02DC 12:        0A20 MPY10:  INC SP  ..RECOVER LOOP COUNT

02DD 9F:        0A21      GHI AC  ..ADD HIGH BYTE
02DE F4:        0A22      ADD     ..ADD 8ACC TO 2ACC
02DF BF:        0A23      PHI AC
02E0 33FF:      0A24      RDE OVFLW-#01 ..RESULT OF IF OVFLOW
02E2 12:        0A25      INC SP  ..RESET STACK PNTR
02E3 BF:        0A26      GLO AC  ..SAME FOR AC.0
02E4 F4:        0A27      ADD
02E5 AF:        0A28      PLO AC
02E6 9F:        0A29      GHI AC  ..FOR CARRY OUT
02E7 7000:      0A30      ADI #00
02E9 BF:        0A31      PHI AC
02FA 3BAF:      0A32      RNF LOOP ..IF NOT OVFLW, GO BACK FOR MORE
02FC 8A:        0A33      LSKP   ..SKIP STACK RESET
02FD 12:        0A34      INC SP  ..RESET STACK PTR
02FE 12:        0A35      INC SP
02FF 05:        0A36 OVFLW:  SEP RETN ..DF=1
02F0 9F:        0A37 FINAL: GHI AC  ..CHECK IF EXCEED MAX POS NUM LIMIT
02F1 FCB0:      0A38      ADI #80
02F3 3AFC:      0A39      RNZ CP   ..IF NOT POSSIBLE, SKIP
02F5 BF:        0A40      GLO AC
02F6 3AFC:      0A41      RNZ CP   ..IF NOT, GO TO COMP
02F8 9B:        0A42      GHI NR   ..SEE IF IT IS POSITIVE
02F9 FCFB:      0A43      ADI #FF  ..SET DF ACCORDINGLY
02FB 05:        0A44      SEP RETN
02FC 33FB:      0A45 CP:    RDE *-#01 ..OVERFLOWED!
02FE 9B:        0A46      GHI NR   ..TEST FOR SIGN
02FF 3A09:      0A47      RNZ EXIT  ..IF POS, DONE
0301 BF:        0A48      GLO AC  ..IF NEG, SUBTRACT FROM 0
0302 FD00:      0A49      SDI #00
0304 AF:        0A50      PLO AC
0305 9F:        0A51      GHI AC
0306 :          0A52
0306 7D00:      0A53      SDRI #00
0308 BF:        0A54      PHI AC
0309 :          0A55
0309 :          0A56      .. BINARY TO DECIMAL CONVERSION
0309 :          0A57      .. ***DECIMAL NUMBER = AC
0309 :          0A58      .. DECIMAL NUMBER = SIGN,NN,.....,N1,N0
0309 :          0A59      .. SIGN=#0B +
0309 :          0A60      .. SIGN=#0D -
0309 :          0A61      .. N0=10**0 DIGIT
0309 :          0A62      .. N1=10**1 DIGIT, ETC
0309 :          0A63      .. ***** (TO CALL, WRITE) *****
0309 :          0A64      .. ***** CALL CRD: ,A(NUMBER); ,LENGTH
0309 05:        0A65 EXIT:  SEP RETN
030A 46:        0A66 CRD:  LDA LINK ..GET THE ADDRESS
030B BA:        0A67      PHI AR   ..AND STORE IN RA
030C 46:        0A68      LDA LINK ..SAME FOR LOW BYTE
030D AA:        0A69      PLO AR
030F 46:        0A70      LDA LINK ..GET LENGTH
030F FF01:      0A71      SMI #01  ..SUBTRAC FOR SIGN BYTE
0311 AB:        0A72      PLO NR   ..STORE IN NR.0
0312 BB:        0A73      PHI NR   ..AND NR.1
0313 F80F:      0A74      LDI #0F  ..NUM OF ITERATIONS
0315 AD:        0A75      PLO MA   ..STORE IN MA.0
0316 9F:        0A76      GHI AC  ..TEST FOR SIGN
0317 FF:        0A77      SHL

```

```

0318 F80B:      0878      LDI #0B  .. IF DF=0, IT IS POS
031A 3B29:      0879      BNF POS  ..
031C 8F:        0880      GLO AC  ..OTHERWISE CONVERT IT TO POS
031D FD00:      0881      SDI #00
031F AF:        0882      PLO AC
0320 9F:        0883      GHI AC
0321 7D00:      0884      SDBI #00
0323 RF:        0885      PHI AC
0324 F80D:      0886      LDI #0D  ..MINUS SIGN
0326 C8:        0887      LSKP  ..
0327 F80D:      0888      LDI #0D
0329 5A:        0889 POS:   STR AR  ..PUT IT IN SIGN BYT
032A 8B:        0890      GLO NR  ..CHECK DIGIT COUNTER
032B 3232:      0891      RZ LOOP1-#02  ..GO BACK FOR MORE ITERATION
032D 1A:        0892      INC AR  ..GO TO NEXT DIGIT
032E 2B:        0893      DEC NR  ..DEC DIGIT CNTR
032F 3027:      0894      BR POS-#02  ..GO BACK FOR MORE CLEAR
0331 F2:        0895      SEFX SP
0332 9B:        0896      GHI NR  ..RESET DIGIT CNTR
0333 AB:        0897      PLO NR  ..
0334 BF:        0898 LOOP1:  GLO AC  ..SHIFT BIT OF AC OUT
0335 FF:        0899      SHL  ..
0336 AF:        0900      PLO AC
0337 9F:        0901      GHI AC
0338 7F:        0902      SHLC  .. SAME FOR AC.1
0339 BF:        0903      PHI AC
033A 0A:        0904      LDN AR
033B 7C00:      0905      ADCI #00  ..ADD TO LOWEST DIGIT
033D 5A:        0906      STR AR
033E 8D:        0907      GLO MA  ..FOR MORE ITERATION?
033F 3A42:      0908      RNZ *#03  ..CONTINUE IF MORE ITERATION
0341 D5:        0909      END:   SEP RETN
0342 0A:        0910 NEXT:  LDN AR  ..LOAD DIGIT
0343 7F:        0911      SHLC  ..SHIFT LEFT OVER ONCE
0344 5A:        0912      STR AR  ..PUT IT BACK
0345 FF0A:      0913      SMI #0A  ..NEED TO INC NEXT DIGIT?
0347 3B4A:      0914      BNF *#03  ..SKIP IF NOT>10
0349 5A:        0915      STR AR  ..ELSE UPDATE DIGIT
034A 2A:        0916      DEC AR  ..GO TO NEXT DIGIT
034B 2B:        0917      DEC NR  ..DEC DIGIT COUNT
034C AB:        0918      GLO NR  ..CHECK IF 0?
034D 3A42:      0919      RNZ NEXT  ..IF NOT, DO THE SAME FOR NEXT DIGIT
034F 3341:      0920      RDF END  ..OVERFLOWED
0351 2D:        0921      DEC MA  ..DEC NO OF SHIFTS
0352 9B:        0922      GHI NR  ..RESET ADDRESS PTR
0353 52:        0923      STR SP  ..PUT DIGIT ON STACK
0354 BA:        0924      GLO AR
0355 F4:        0925      ADD
0356 AA:        0926      PLO AR
0357 9A:        0927      GHI AR
0358 7C00:      0928      ADCI #00
035A BA:        0929      PHI AR
035B 3032:      0930      BR LOOP1-#02  ...
035D :          0931      END

```


Appendix C - Sample Program

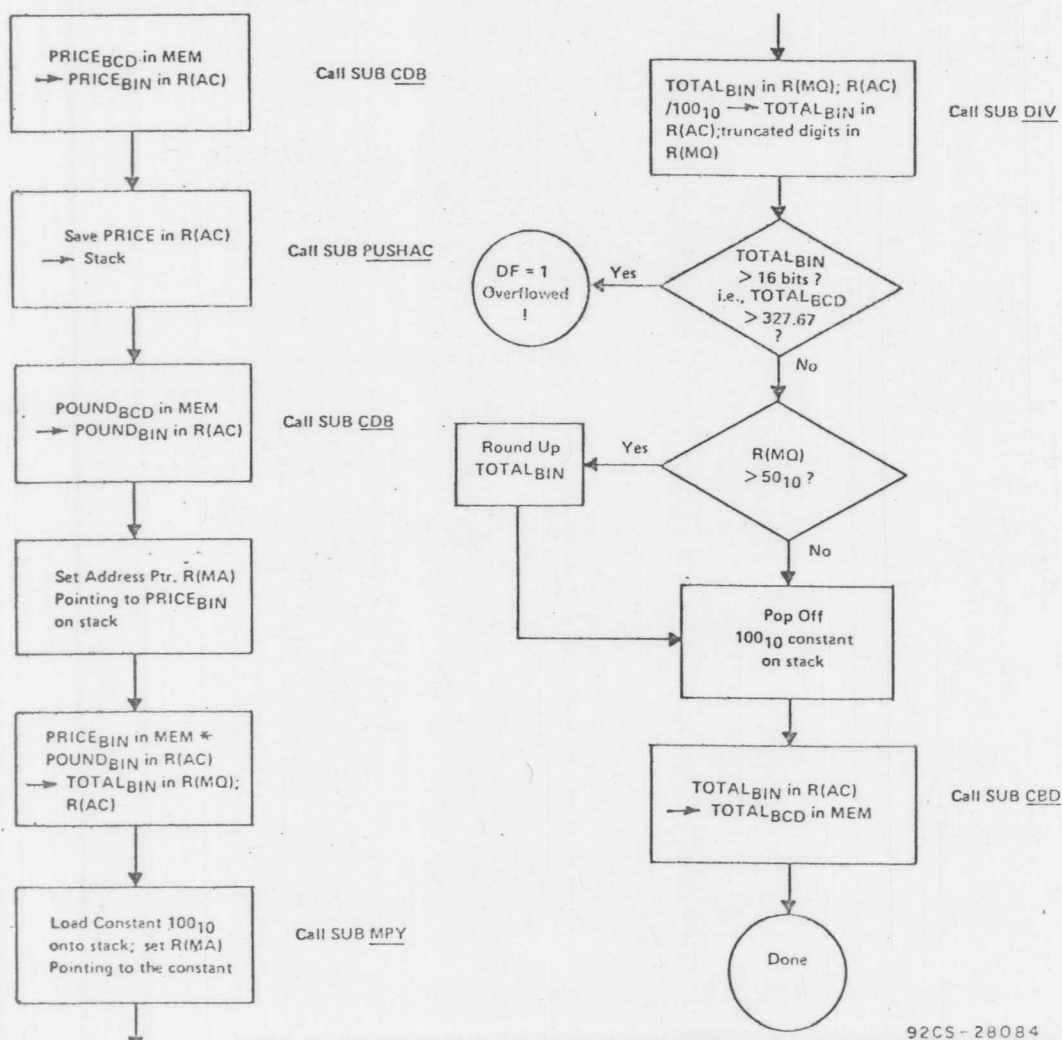
A sample program to demonstrate the use of the Arithmetic Subroutine Package is given in this appendix. This program accepts two input parameters—PRICE and POUND, calculates the PRICE·POUND product (TOTAL), and outputs TOTAL after rounding and truncation.

The input parameters are in BCD form and a maximum of four digits is allowed. A decimal point is also assumed on the left of the second least significant digit, i.e., $00.00 \leq \text{PRICE}, \text{POUND} \leq 99.99$.

These parameters are then converted into binary numbers and TOTAL is calculated by multiplying PRICE

and POUND in binary. This binary product is divided by 100 to eliminate the last two decimal digits of the eventual BCD TOTAL. If the remainder of the division is greater than fifty, one is added to the 16-bit quotient (for rounding purposes). This latter is then converted back into BCD form.

Note that since the binary-to-BCD routine is designed for 16-bit conversion only, an overflow condition will occur if TOTAL is greater than 327.67. The DF flag sets if overflow occurs.



Flow chart of sample program.

92CS-28084

0215 ;	0037	
0215 D4;	0038	SEP CALL .. DO DECIMAL TO BINARY CONVERSION
0216 076E;	0039	,A(CDB) .. "
0218 027D;	0040	,A(PRICE) .. CONVERT PRICE INTO BINARY
021A 05;	0041	#05 .. PRICE IS 5 CHARS LONG
021B ;	0042	... AC NOW CONTAINS BINARY VALUE
021B ;	0043	.. OF PRICE PER POUND
021B ;	0044	
021B D4;	0045	SEP CALL .. PUSH CONTENT OF AC INTO STACK
021C 06E9;	0046	,A(PUSHAC) .. "
021E ;	0047	
021E D4;	0048	SEP CALL .. DO DECIMAL TO BINARY CONVERSION
021F 076E;	0049	,A(CDB) .. "
0221 0282;	0050	,A(LBS) .. CONVERT QUANTITY (LBS)
0223 ;	0051	... INTO BINARY
0223 05;	0052	#05 .. QUANTITY IS 5 CHARS LONG
0224 ;	0053	.. AC NOW CONTAINS BINARY VALUE
0224 ;	0054	.. OF QUANTITY (LBS)
0224 ;	0055	
0224 82;	0056	GLO SP .. COPY STACK POINTER TO MA
0225 AD;	0057	PLO MA .. "
0226 1D;	0058	INC MA .. POINT TO AC. 1
0227 ;	0059	
0227 D4;	0060	SEP CALL .. DO THE MULTIPLICATION
0228 0475;	0061	,A(MPY) .. "
022A ;	0062	
022A ;	0063	... DIVIDE BY 100 TO REMOVE LAST TWO
022A ;	0064	... DECIMAL DIGITS
022A 12;	0065	INC SP .. MOVE SP TWO BYTES
022B 12;	0066	INC SP .. BELOW TOP OF STACK, POP PRICE
022C ;	0067	... OFF STACK
022C F864;	0068	LDI 100 .. LOAD 100 INTO STACK WITH
022E 52;	0069	STR SP .. SP POINTING TO THE HIGH BYTE
022F 22;	0070	DEC SP .. "
0230 F800;	0071	LDI 00 .. "
0232 52;	0072	STR SP .. "
0233 82;	0073	GLO SP .. COPY STACK POINTER TO MA
0234 AD;	0074	PLO MA .. POINT TO HIGH BYTE OF 100
0235 22;	0075	DEC SP .. POINT TO FREE SPACE
0236 D4;	0076	SEP CALL
0237 051E;	0077	,A(DIV) .. DIVIDE PRODUCT BY 100
0239 334A;	0078	BDF LAB3 .. IF OVERFLOW GO TO LAB3
023B ;	0079	
023B ;	0080	... CHECK IF REMAINDER IS GREATER
023B ;	0081	... THAN 50, IF SO, ROUND UP
023B 8E;	0082	GLO MQ .. MQ CONTAINS THE REMAINDER
023C FF32;	0083	SMI 50
023E 3B41;	0084	BNF LAB1 .. IF NO ROUND UP
0240 ;	0085	... GO TO LAB1
0240 1F;	0086	INC AC .. IF ROUND UP ADD 1 TO
0241 ;	0087	... THE LEAST SIGNIFICANT DIGIT
0241 ;	0088	
0241 12;	0089 LAB1:	INC SP .. MOVE SP DOWN TWO BYTES
0242 12;	0090	INC SP .. BELOW TOP OF STACK
0243 ;	0091	
0243 D4;	0092	SEP CALL .. DO BINARY TO DECIMAL CONVERSION
0244 0645;	0093	,A(CBD) .. "
0246 0287;	0094	,A(TPR) .. CONVERT TOTAL PRICE INTO DECIMAL
0248 76;	0095	#06 .. TOTAL PRICE IS 6 CHARS LONG
0249 ;	0096	.. TOTAL PRICE IS STORED IN M(TPR)
0249 ;	0097	

Partial assembly-language listing of the calculation subroutine.